```
          =
        IUCMS
        ==  =
        =    =
        =     =
        =   =
        =  ==
        = ==
        ===
        ==
       ==
      == =
    ===  =
   ===  MUSIC==
  ==   IUCMSIUCMS
 ===  == =     ===
 ==  ==  =      ===
=== ==   =       ==
 == ==   =       ==
 ==   =  =      ==
  ==   = =     ==
    MUSICMUSIC==
      IUCMS=
         =
         =
     ==  =
    ====  =
    MUSIC=
     ====
```

Music Notation by Computer

by

Donald Alvin Byrd

Submitted to the faculty of the Graduate School in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science, Indiana University

August, 1984

Accepted by the faculty of the Computer Science Department, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee: Douglas R. Hofstadter, Chairman
Douglas R. Hofstadter

J A Barnden
John Barnden

John O'Donnell
John O'Donnell

Mitchell Wand
Mitchell Wand

C. Allen Winold
C. Allen Winold (Minor Field Advisor)

May 24, 1983

Abstract

Music Notation by Computer
Donald Alvin Byrd

Musicians made their first attempts to use computers for musical purposes about 25 years ago. A major problem that confronted them was that neither musical sound nor the standard representation of music — conventional music notation (henceforth called "CMN") — could be either input or output by existing computer systems. Since then all four aspects of the musicians' problem — sound input, sound output, CMN input, and CMN output — have been attacked, with varying degrees of success. This project is concerned with one of these aspects, namely CMN output.

Producing CMN output is most generally viewed as having three aspects: *selecting* the symbols to print, *positioning* them, i.e., deciding where to print them, and actually *printing* them. The positioning aspect is the core of the problem: it is essentially a question of formatting, analogous to formatting natural language text, but much more difficult. A solution to this aspect of the musicians' problem, that of automatic music formatting, would not only be very valuable to musicians, but would also be quite interesting and nontrivial from the computer scientist's viewpoint.

I argue, and give examples by major composers to show, that "fully automatic high-quality music notation" is not merely nontrivial but in general impossible without human-level intelligence. Since current work in artificial intelligence is far from this level, one must compromise, and three ways are possible: compromise in degree of automation, in quality of output, or in generality of input. Most workers on the problem have primarily sacrificed automation. The present work, in contrast, primarily sacrifices generality of input, partly due to historical accident but partly on pLilosophical grounds. A major chapter of the dissertation documents my program as so far implemented. The dissertation concludes with a discussion of the central unsolved problems of automatic music formatting.

Table of Contents

List of Figures

CHAPTER 2

CHAPTER 3

CHAPTER 4

CHAPTER 5

This book is dedicated to my father and to the memory of my mother,
about whom nothing can be said.

Acknowledgements

The research described herein may or may not have required more effort than the average Ph.D. dissertation, but I'm virtually certain that I got much more help from many more people than the average Ph.D. student does. This research has involved two fairly distinct phases: developing my music-setting program, SMUT, from 1968 to the present, April 1983 (on two successive CDC systems, a 3600 and a 6600, at Indiana University's Wrubel Computing Center); and writing this dissertation (on a VAX 11/780 under UNIX, at I. U.'s Computer Science Department), over the last year or so. I want to thank some people for help on one phase, some for help on the other, and some for help on both. And I want to thank many people for being my friends (I will not say "just" for being my friends).

My mother, Charlotte Klein Byrd, and my father, Eugene Byrd, have been the most important people in my life. My statement about them appears on p. xii.

For advice on matters of large scale or small, I have turned repeatedly to Douglas Hofstadter. On the large scale, Doug has an unerring sense of what is most important in an amazing number of things, including computer setting of music, and the documentation thereof. On the small scale, he has an eye and a concern that I can only call extraordinary for such details as grammar, word choice, spelling, and even typography. If my dissertation *looks* good, to Doug is due much of the credit. Of importance to me on all levels is Doug's rare quality: his sense of style. My only fear is that I won't be able to learn as much from him in the future, once the sheepskin is awarded. But Doug's influence is spread throughout, which pleases me greatly; it can be seen quite clearly in this very sentence, not to mention five of the previous six. It can also be seen in many of the paragraphs, if not sentences, after this.

Jim Hettmer has probably had more influence on me professionally than anyone. In the dozen or more years I have known him and the perhaps seven years we worked together at the Wrubel Computing Center, he helped me learn more than I can easily say about how to design programs that stand in the user's way as little as possible, about computer graphics, and about — well, about *computers*.

Composers usually care more about music notation than other musicians, and Jeffrey Hass is exceptional even among composers in his concern for it. Jeff has for some time been an enthusiastic user of the music setting system described herein. He's also spent many hours reading drafts of parts of this dissertation, and given me many valuable suggestions about both the programs and the dissertation. The moral support he has given me is less easy to put into words, but is no less important to me — perhaps more.

Steven Johnson has helped me a great deal (as even he admits) with both the content and the form of this dissertation. He has helped me with the content through numerous conversations which have significantly affected my attitude about music setting as an application of computers, and with the form by helping me with a program that stands in its users' way quite a bit, namely the UNIX TROFF typesetting system.

John O'Donnell has gone beyond the call of duty by spending time listening to me flounder verbally when I did not know how to do anything else. I do not mean to suggest that he was not helpful when I was coherent: he certainly was, and on many occasions. He also brought to my attention the paper by Kirkpatrick et al discussed in Sec. 5.2.2.

Over the years, two members of the I.U. Music Theory Department, Allen Winold and Gary Wittlich, have given me the viewpoint of music theorists with exceptional sympathy for and understanding of computers — and very high standards, as well. In addition, I got my very first formal training in programming in Gary's class "Introduction to Computer-Aided Music Research" quite some time ago (SNOBOL3 on a CDC 3600, if you're curious), and students of Gary's have unearthed many a problem with my system.

Absolutely nobody has influenced my overall outlook more than my twin brother, Richard Byrd. One small example: he was the person who first introduced me to computers. More specific to this project, I am indebted to him for some illuminating comments on mathematical notation and its relation to music notation.

David DeLauter gave me my first job as a programmer in 1969. We became very close friends, and he influenced my life deeply in many ways, every one for the good.

Probably the only person I've ever met who is as fascinated with notation systems as I am is Scott Kim. As evidence, let me quote his marginal comment on an early draft of what is now Sec. 2.5: "What fun." He gave me, several years ago, a crucial insight into rhythm notation which is incorporated in the rhythm clarification algorithm I describe. He also made many perceptive comments on Chapter 2.

George Alexander, an expert on printing systems, gave me quite a bit of advice (nearly all of which I have taken) on several topics, especially phototypesetters.

My music printing program, SMUT, is dedicated to Iannis Xenakis, with and for whom I worked some 12 years ago. I want to thank him for his unique and wonderful music as much as for what he helped me learn.

Rosalee Nerheim did a tremendous amount of work on various parts of the Indiana University Computer Music System (IUCMS), especially those related to music printing. She has also given me an amazing amount of support and encouragement over the years.

Bruce Rogers was perhaps the first composer to take SMUT seriously as a tool for his own use (this was in 1974). His interest was very important to me. He also made

significant contributions as a programmer to the IUCMS, as did George Cohn and Gary Levenberg.

My office mate, Scott Jones, has been as considerate as I could have wished for — in fact, more considerate: he's done things to make life easy on me I would not have thought of asking for. Also, in his role as Computer Science Department systems programmer, he's done a fine job of keeping things running smoothly.

Thanks to John Barnden and Mitchell Wand for serving on my research committee; to Kimball Stickney for converting SMUT to run under VAX/VMS and for some last-minute logistic support; to Jim Emery for sharing with me his bibliography on computer music printing; to Geoffrey Hellman for advice on notation from the (musical) logician's viewpoint; to Bernie Greenberg for comments on some fine points of music notation and for several examples; to Fanya Montalvo for advice on user interface and artificial intelligence questions and for general encouragement; to Being David Moser for being David Moser, something more valuable to me than I can say. Thanks to the experts who've shared their knowledge with me via "personal communication": Lawrence Bodony, Armando Dal Molin, Cameron Jones, William Kornfeld, James A. Moorer, Severo Ornstein, Leland Smith, Laurie Spiegel, Rochelle Vickey, William Watkins, and Robert Webber. Thanks to Paul Glenn, Frank Prosser, and Steve Young for creating the incredibly pleasant working environment of the Wrubel Computing Center, now dead.

Finally, for hundreds of reasons, I want to thank Wilson Allen, Leonard Bernstein, Barbara Bialick, Tim Bomgardner, Nancy Burnham, Bill Buxton, David Byrd, Bob Chidlaw, Mark Curry, Susan Dailey, Nancy DeLauter, Jim Dublin, Mark Fasman, Bob Filman, Jennifer Frick, Adrienne Gnidec, Tulle Hazelrigg, Mike Hosek, Greg Kendall, Dave Kriewall, Jan LaRue, Margot Leverett, Bill Lozowski, Nikki Martin, John Maurer, Lyle Mays, Dennis McGreer, Larry Meehan, Paul Nadler, Dwight Noble, Greg O'Drobinak, Tresina Pavel, Charles Pearce, M. F. Suzanne Pinette, C. Roads, Bill Robbins, Joyce Robbins, David Roth, Ann Segraves, Tim Sevener, Michael Shoemaker, Kate Sillars, Pat Simpson, Laura Stone, Robert Weisz, Jerome Wenker, and Loren Weybright.

D. B.

Preface

I have just a few general comments to make here.

First, even though this dissertation is "in" computer science, its content deals to a great extent with musical matters, and I want to make clear how seriously I take the musical content. In the Preface to *The Art of Computer Programming* Donald Knuth remarks: "Since I myself profess to be a mathematician, it is my duty to maintain mathematical integrity as well as I can." I profess to be a musician, and I consider it my duty to maintain musical integrity, not "as well as I can", but simply and without qualification "well".

Second, since this dissertation is concerned with formatting and printing, readers may be especially interested in how it was itself prepared, both music and text. The text was produced entirely on a VAX 11/780 under Berkeley UNIX. It was entered and edited with EMACS; set in Computer Modern with ITROFF, a version of the well-known TROFF typesetting program; and printed on a Canon/Imagen LBP-10 laser printer with 240 points per inch resolution. The examples of mathematical notation in Chapter 2 were also set by ITROFF, aided by the EQN mathematics preprocessor. In Chapter 5, Fig. 2 was set by ITROFF, and Fig. 4 is from G. M. Hunter's dissertation [HUNT78]. The musical examples were prepared in several different ways. Some were set by my MUSTRAN/SMUT music printing system running on a CDC 6600 and printed on a Zeta 3653SX pen plotter or Versatec 1200 electrostatic plotter; the captions of figures set this way explicitly identify them as such. The others were either reproduced from published editions or hand-drawn, and the reader should have no difficulty telling which are which. All other figures were drawn by me. Figure captions and text within figures were prepared in several different ways, which I will not go into.

The body of this dissertation was completed over a year ago, in mid-1983. I have added a postscript to cover developments through mid-1984.

It is probably pointless to carry out the listing of these horrors any further. By now the reader can likely begin to construct his own examples of barely manageable metric complexity. It should certainly be clear that if there is any sort of general solution it is not one that jumps forth and makes itself unmistakably known.

David Gomberg, *A Computer-Oriented System for Music Printing* (Sc.D. dissertation, Washington University, 1975), p. 64

The interesting point to notice is the admirable illustration which this [Arabic] numeral system affords of the enormous importance of good notation. By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race. Before the introduction of the Arabic notation, multiplication was difficult, and the division even of integers called into play the highest mathematical faculties. Probably nothing in the modern world would have more astonished a Greek mathematician than to learn that, under the influence of compulsory education, a large proportion of the population of Western Europe could perform the operation of division for the largest numbers.

A. N. Whitehead, *An Introduction to Mathematics* (Oxford, 1911)

Standard music notation offers a familiar and at the same time a remarkable case [of a notation system actually used in the arts]. It is at once complex, serviceable, and — like Arabic numerical notation — common to the users of many different verbal languages. No alternative has gained any currency; and apparently no other culture, such as the Chinese or the Indian, has developed any comparably effective music notation over the centuries.

Nelson Goodman, *Languages of Art* (Hackett, 1976), p. 179

Computer music systems must continue to evolve in order for their developers to incorporate all of the refinements that will make them truly efficient and "friendly" tools for musicians. After all, the violin evolved over centuries and the acoustic piano took generations to reach its present musical excellence. We have no reason to believe that the ultimate computer music system, whether it costs $1000 or $50,000, is close at hand.

Robert A. Moog, "The Soundchaser Computer Music System", *Byte* 7,12 (December 1982), p. 276

The literary arts use a basic set of symbols, the Roman alphabet, which has taken some 3,000 years to develop and has ousted several elaborate systems along the way. The musical arts use an entirely different system, staff notation; and though its period of continuous development has been shorter — perhaps no more than 1,500 years — its task is very much greater.

Thurston Dart, *The Interpretation of Music* (Harper and Row, 1963), p. 11

By now it is surely evident that the notator must approach his decisions with a flexible mind.

Gardner Read, *Music Notation*, 2nd ed. (Crescendo, 1969), p. 86

1
Introduction

1.1. THE MUSIC SETTING PROBLEM

Musicians made their first attempts to use computers for musical purposes about 25 years ago.[1] A major problem that confronted them was that neither musical sound nor the standard representation of music, conventional music notation (henceforth called "CMN"), could be either input or output by existing computer systems. Since then all four aspects of the musicians' problem — sound input, sound output, CMN input, and CMN output — have been attacked, with varying degrees of success. This project is concerned with one of these aspects, namely CMN output.

Producing CMN output is most generally viewed as having three aspects: *selecting* the symbols to print, *positioning* them, i.e., deciding where to print them, and actually *printing* them. The positioning aspect is the core of the problem: it is essentially a question of formatting, analogous to formatting natural language text but much more difficult. A solution to this aspect of the musicians' problem, that of automatic music formatting, would not only be very valuable to musicians, but would also be quite interesting and nontrivial from the computer scientist's viewpoint.[2] It is so far from trivial, in fact, that most workers on the problem have abandoned totally automatic formatting and developed systems with important interactive components. Such an approach has substantial advantages quite apart from sidestepping intractable problems. The approach has been very successful, but has lead to interesting and difficult questions of user interface design. The present approach, in contrast, is highly automatic, partly due to historical accident but partly on philosophical grounds.

1.2. TERMINOLOGY

We will need a small amount of terminology before going on. The first term that needs clarification, in fact, is in the title of this dissertation. "Music Notation by Computer" is a somewhat misleading or, at best, ambiguous title. I might have said

"Music Printing by Computer", but "printing" is not really the right term, since it suggests reproduction processes as well as the process of preparing the master copy. To clarify the problem and the usage I will adopt, I cannot do better than to quote the one previous computer science dissertation on music notation by computer [GOMB75, p. 1 – 2]:

> The term *music printing* encompasses several production processes. An engraver (or other craftsman) produces an original image of the eventual printed page. To preserve the original image, it is replicated several times and the actual printed impressions are made from the replicas. There is no simple generic term to describe the production of the original image because the method of production usually lends its name to the process. The terms *set* and *setting* in conjunction with printing generally imply the use of type slugs, and although there have been some attempts to use typesetting for music, these terms will be adopted here to describe the production of the original image regardless of the method of production. Music *setting* then is to be understood as the process of placing music symbols in their required places. Music *printing* will refer to the entire set of processes involved in producing a finished piece of printed music.

Strictly speaking, then, my title should be "Music Setting by Computer", but the phrase "music setting" is so little used that it might have caused worse misunderstandings than the less accurate title.

To put things in context more, two related terms are "transcribing" and "amanuensis". Musicians often use the term *transcribing* for a process that produces music notation, starting either with sound or with notation for a different combination of instruments and voices; either process includes music setting as a subprocess. Transcribing music (in the first sense of "transcribing") by computer is discussed in Sec. 3.2.2. An *amanuensis* is one who listens to a performance and writes it down in music notation, as in Jef Raskin's "Using the Computer as a Musician's Amanuensis" [RASK80].

For definitions of other terms, see the Index to Definitions.

In summary, this dissertation is concerned primarily with music setting. I compare terminology in several other fields to that used in music in Sec. 5.3.

1.3. PROBLEMS AND GOALS

Music notation is like the written notation of virtually every natural language in that it was not designed by any specific person or group of people; rather, it evolved over a period of centuries. In the case of CMN, this evolution took place mostly in Europe between about 1000 and 1600 A.D., although a small amount of further change has occurred continuously since then. Although CMN originated in a manner similar to natural language notations and has many structural similarities to them, it is quite different in not being based on any fixed character set, no matter how large, and in relying much more heavily on symbol position to convey information. This is an indication of the enormous complexity of CMN[3], and is a reason why a music-notation formatting program is fundamentally different from, and much more difficult than, a text-formatting program. In a word, CMN is essentially graphical; written natural language is not.

Thus, a music notation system has immediate interest from a computer science standpoint because it attacks a highly complex graphics problem — in fact, one of a complexity that has never been handled very satisfactorily by computer alone, that is, without a substantial amount of manual intervention.[4] This is certainly true, because much music exists whose correct formatting requires considerable intelligence (well beyond the state of the art of artificial intelligence), and some music exists whose correct formatting probably requires full human intelligence. These are strong and obviously unprovable assertions, but I will give some arguments and many examples to support them (in Chapter 2).

An interesting graphics problem that occurs in music notation and that has definite artificial intelligence implications is that of placing symbols to avoid collisions. How can it be insured that slurs (for example) do not run into noteheads? The problem is greatly complicated by the fact that some collisions are perfectly acceptable: for example, published music frequently has slurs intersecting barlines or accidentals (see Sec. 2.5). Even if one can detect objectionable collisions, what should be done when one is found? What symbol or symbols should be moved, and where to? Similar situations can occur in other graphics domains, for example in cartography (labelling map features) and electronic design (positioning components and paths). Substantial work has been done on this problem in both domains; however, little if any is general enough to apply to music notation. This is not surprising because any truly general solution to this problem would really be a solution to what Hofstadter has called [HOFS83a] "the slippability problem" of artificial intelligence. I discuss collision avoidance in Sec. 5.2.1.

Thus, one of the main goals of this work is to find usable approximate solutions to problems in graphics whose complete solution is certainly many years, probably decades, off. Additional goals are:

(1) To gather explicit knowledge about one of the most sophisticated notational systems in existence, namely CMN. There are of course many books on music notation, but all, or nearly all, were written by musicians for musicians, and the natural result is that they make enormous assumptions.[5] A related problem with all of the books I am familiar with is that they sometimes give incorrect rules in conjunction with correct examples, apparently as a result of generalizing improperly. We will see an example of this in Chapter 4.

(2) To investigate program portability and user interface problems in a graphics program of significant complexity.

The analogy between written language and CMN made above is an extremely useful one, and I will return to it repeatedly throughout this dissertation. Natural language formatting is a process most computer scientists are at least somewhat familiar with; an enormous amount of study has been devoted to it; and nearly all of the problems it involves occur in music formatting (along with numerous others). Furthermore, this analogy has barely been hinted at in the existing literature on music setting, either by hand or by computer. I will particularly rely on two well-known text-formatting systems for examples: Knuth's TEX, primarily because its underlying philosophy is exceptionally general and so it provides far more interesting ideas than do most text formatters, and the UNIX TROFF system, simply because this dissertation was formatted by it and I am more familiar with it than any other system.

A useful discussion of many aspects of computer setting of music, including the history of music printing and several purely graphical problems, appears in David Gomberg's dissertation [GOMB75]. Gomberg also does some philosophizing on the relative merits of batch and interactive approaches, from which I quote in Sec. 3.4.

1.3.1. This Work Related to Artificial Intelligence

The relationship between this work and artificial intelligence (AI) is worth commenting on. A major point of this dissertation, made especially in Sec. 2.5, is that programming "Fully Automatic High-Quality Music Notation" (henceforth, "FAHQMN") is a research problem in AI, in fact one that is well beyond the state of the art; for precisely this reason my program, SMUT, is not an AI program. I wanted a program of practical value and felt that I could write one only by eschewing AI techniques and not attempting to accomplish true FAHQMN. Let us consider what the possible ways are of approximating FAHQMN.

The wording of the phrase "Fully Automatic High-Quality Music Notation" suggests three possible types of compromise:

(1) Not being Fully Automatic, i.e., the user must tell the system more than the bare minimum that one would need to tell an expert human music engraver. For example, an experienced engraver can decide correctly in almost all cases what notes to beam together (see Sec. 2.3.3.2.2) and where to position the beams; however, doing so sometimes requires (as I have already suggested) considerable understanding of music, so no program in existence can do as well as a human expert. In fact, some systems never beam notes together at all unless they are explicitly told where each beam should begin and end. A variation on this is Xerox PARC's Mockingbird (Sec. 3.4), which generates a non-standard time notation that does not use beams at all; however, it provides powerful interactive tools for converting this notation to CMN. The level of automation of a system is, admittedly, a rather fuzzy concept: How automatic is a system that decides automatically what notes to beam together and where to position the beams; that makes frequent and elementary mistakes; and that allows overriding its decisions? What if it provides no method for overriding its decisions? What if it makes only occasional, relatively sophisticated mistakes? This question is also related to the batch/interactive dichotomy. One might argue that a batch system that allows overriding of its decisions is *de facto* more automatic than an otherwise identical interactive system that allows overriding its decisions, since the manual override will be far less convenient and hence far less often used in the batch case.

(2) Not being High Quality. Some aspects of high quality in music notation are very straightforward, e.g., the shapes of "fixed" symbols such as clefs and accidentals. Others are less obvious and interact with point 1, Full Automation. For example, a system might — left to itself — draw a slur that would be fine in another context but that intersects note stems unnecessarily. If this system is not fully automatic and allows changing the shape and/or position of the slur, it is still capable of high quality.

(3) Not covering all of Music Notation, i.e., handling only a subset of music notation. In this thesis I consider only CMN, which — extensive though it is — already excludes a great amount of music notation in the broadest sense. Also, as we will see in Secs. 2.3.6 and 5.2.3, most of the really hard problems of music notation are exacerbated by complexity of "texture", that is, by the presence of multiple independent voices on a staff (see Sec. 2.3.6 for a definition of "texture" and Secs. 2.3.1 and 2.3.2 for definitions of "voice" and "staff"). As a result, few systems can handle more than two independent voices on a staff. Again, interactions among the compromises exist: no system I know of handles more than two independent voices per staff *automatically.*

In my opinion, every existing CMN setting system, including mine, compromises on not one but at least two of the three aspects — in most cases with the author's conscious intent, in some without. As I have already suggested, most systems compromise the most on point 1, Full Automation, while my system compromises primarily on point 3. See also Sec. 5.2.1.4.

1.4. METHODOLOGY AND SCOPE

The main methodology of the research was to write a program:

(1) Capable in design of handling virtually all Western music (art or otherwise) written from about 1600 to 1945, and in implementation of handling, in art music of about 1700 to 1935, nearly all wind music, all string music that does not involve "multiple stops" (two or more simultaneous notes), texturally simple keyboard music (e.g. Bach Two- and Three-Part Inventions, some of Bartók's *Mikrokosmos* but almost no Chopin), and many orchestral scores (limited in number of voices only by memory and plotter capability). More specifically, the

implementation is able to handle one or two rhythmically independent lines on a staff with each line having only one note at a time, and any number of staves (again limited by computer memory). (Much string music has occasional chords of two to four notes. Of course, most piano music has chords of up to five notes. The Bach *Chaconne* for unaccompanied violin has, in places, three rhythmically independent lines; piano music such as Chopin's sometimes has this feature and often has lines moving from one staff to another.)

(2) Able to produce publishable and nearly "publication" (i.e., typical engraved music) quality.

(3) Portable from computer to computer, including minicomputers and large microcomputers, by virtue of an appropriate programming language and style. For several reasons including that of paragraph 6 below, I used FORTRAN IV.

(4) Portable to any reasonable graphic output device, either raster or random scan. This was accomplished (though at considerable cost in efficiency and programming effort) by assuming a minimal set of primitives, specifically those which: draw a vector, write a string of characters in an undefined font, and convert a number to a string of characters and write it.

(5) Efficient (in spite of paragraphs 3 and 4) and reliable enough for practical use.

(6) Integrated with the existing Indiana University Computer Music System [BYRD77a] to facilitate practical use.

This program, named SMUT (System for MUsic Transcription), is batch-oriented, not interactive. It is clear to me that a really useful music notation system needs to be interactive; however, it also needs *all* of the capabilities of a batch system, so that I feel essentially none of my work has been wasted. This relates to the question of "levels of control" in a system's user interface, which I feel has been neglected by most system designers; see Secs. 5.2.3 and 5.3.1.

1.5. OVERVIEW

The remainder of the dissertation is organized as follows.

Chapter 2, "Notation Systems and Music Notation", has three main sections. The first, "Conventional Music Notation for the Computer Scientist", is primarily intended to give the computer scientist who knows little or nothing about CMN enough background to understand the remainder of this and following chapters. It should also be of interest to persons looking for a description of music notation that is much more systematic than the usual ones, though it is still far from formal in the mathematical sense. The other two sections are "A Comparison of Notation Systems" and "Counterexamples". These are intended to back up my assertions about the complexity of CMN and the intelligence required to produce it correctly. In a sense, these sections provide the justification for this dissertation in that they show how difficult on several different levels music setting is.

Chapter 3 is "Background and Related Work". It discusses the three parts of the entire CMN problem (what to print, where to print it, and printing it) and approaches to CMN output, and summarizes the work of several researchers on automating music notation from the 1950's to the present. It also gives some background for my own work. One of the other aspects of "the musicians' problem" mentioned at the beginning of this chapter is important to CMN output systems, namely CMN input systems; these are discussed in this chapter at some length.

Chapter 4 describes "How SMUT Works". In order to give the reader some intuition about the problem, it begins by pointing out problems in setting several sample excerpts of music of increasing complexity. It then discusses the basic "Principles of Operation" of SMUT's four passes, then goes into detail about some of its more interesting features, especially those relating to rhythm. The chapter closes with further discussion of the setting of one of the sample excerpts.

Finally, Chapter 5 is "Conclusions". Although the combined efforts of many workers have led to considerable progress towards a really useful automatic notation system, music itself (and, therefore, music notation) is much too complex to yield fully to a mere 25 years of work. As I have already pointed out, many of the problems in CMN setting also occur in other domains, so one might instead go back 35 years to the first electronic computers — still not long enough to solve problems of such magnitude. In this chapter I discuss the work that remains to be done. For example, I say quite a bit about the collision-avoidance problem mentioned above, which is very important in several domains. I comment on the difficulty of handling complex textures and its relationship to collision avoidance. I also discuss "Editing and Formatting, Design and Drafting" and the importance and problems of systems that support multiple levels of control.

NOTES

[1] An entire book about computers and music was published as early as 1959: [HILL59].

[2] To my knowledge one previous dissertation in computer science has been devoted to this subject, namely [GOMB75]. Unlike me, Gomberg did not actually implement a music formatting system, although he did analyze many of the problems. There have also been several dissertations and master's theses in both music and computer science devoted to more or less closely related issues, e.g., [PRER71], [KNOW71], [MOOR75], [REND81].

[3] As further evidence, a standard text, [READ69], is 482 pages in length. See also Chapter 2, especially Sections 2.3.6, 2.4, and 2.5.

[4] Several systems (e.g. Armando dal Molin's "Music Reprographics" system [DALM75, DALM78], Leland Smith's MSS [SMIT73, SMIT78]) do quite well either by relying on a human to interactively "guide" the computer, or by requiring manual finishing of the printed music after it has been computer drawn.

[5] The standard texts are [READ69], [ROSS70], and [STON80]. The closest thing I know of to an explicit statement of the rules of music notation is in [ROSS70]. This is still no more than, to choose a number, 5% explicit (as compared to the usual value of, say, 2%).

2
Notation Systems and Music Notation

2.1. INTRODUCTION

Of the many forms of music notation in use around the world, my work is concerned exclusively with the one most familiar to Westerners: the notation developed originally for Western music of the last few hundred years and now in use for many types of music, especially art music. I will call this notation "conventional music notation", or "CMN" for short.

Like the written notation of virtually every natural language, CMN was not designed by any specific person or group of people; rather, it evolved over a period of centuries. In the case of CMN, this evolution took place mostly in Europe between about 1000 and 1600 A.D., although its roots go back far earlier and a small amount of change has occurred continuously since 1600.

Understanding any notation requires an understanding of the information the notation attempts to capture. In the case of CMN, four types of information are involved, namely the four parameters of musical sound (at least according to Western music theory): *pitch, time, loudness,* and *timbre.*[1] The basic principles of the representation are simple. CMN uses a highly modified two-dimensional Cartesian coordinate system to represent the four parameters:

(1) *Pitch* is represented explicitly along the vertical axis, although the representation is nonlinear and complex. It relies heavily on the *staff*, a set of five horizontal lines for reference (see Sec. 2.3.2).

(2) *Time* is represented fairly explicitly along the horizontal axis; this representation is also nonlinear and even more complex. ("Duration" and, especially, "rhythm" are higher-level temporal aspects of music; more will be said about them below. Furthermore, "time" is often used by musicians as a synonym for "meter", but I use it here in its more colloquial sense.)

(3) *Loudness* (closely related to "dynamics") is mostly implicit. When it is given explicitly, it is by means of verbal comments, usually in Italian and often abbreviated, plus a very few symbols.

(4) *Timbre* (identical to "tone quality") is represented in two distinct ways, one mostly explicit, the other largely implicit. The explicit method is used in compositions for several players to specify which notes belong to which player and instrument; this constitutes a partial specification of timbre. This is accomplished (to oversimplify slightly) by having several carefully aligned staves and using the appropriate one for each sound. Selection of a specific timbre from the (usually ill-defined) set available on the specific instrument is mostly governed by unwritten convention. When it is written, it is expressed in very much the same way as control of loudness.

To restate the situation, the horizontal axis represents pitch. A very short vertical axis represents pitch; it is repeated several times, once (more-or-less) for each possible value of "player" in the piece in question. Thus, as Spiegel has pointed out [SPIE83a], three dimensions are squeezed into two in a way that is often used with graphs, by having one two-dimensional graph of variable 1 against variable 2 for each value of variable 3. Additional loudness and timbre information is given verbally, and still more is implicit.

The above list gives the parameters in roughly decreasing order of importance in Western music theory. It is not surprising that this is also a decreasing ordering of the explicitness with which the parameters are represented in music notation.[2]

A major complicating factor in CMN is that most Western music consists of several synchronized lines going on simultaneously. This would be only a minor complication if the lines were notationally independent and merely needed to be aligned, as with a set of simultaneous equations; in fact, however, they can interact in rather complex ways (cf. Secs. 2.3.6 and 2.4).

If CMN did not concern itself with communicating this rather large amount of information in an efficient, i.e., "high-bandwidth", way, its structure could still be relatively simple. However, CMN is extremely concerned with high-bandwidth communication, and in particular high-bandwidth communication of music written in a particular style: it is optimized for music in approximately the style of Mozart or Beethoven, i.e., typical Western art music of the last half of the 18th or first half of the 19th century. Gomberg makes some perceptive comments on bandwidth in CMN and its relationship to standardization [GOMB75, p. 7]:

> Professional musicians usually cannot enumerate [the conventions of CMN], but they are well aware of them through experience. The basic reason that music setting must abide by these conventions is that music is (often) read in performance at a rate much faster than most persons read ordinary text, and any deviation from the conventional causes the eye to hesitate and thus affects the ability of the musician to perform ... The situation is aggravated by the short rehearsal time available in most ... orchestras. Orchestra players do not attempt to memorize their parts, but instead read it during a performance. Thus it is imperative that music is printed in the manner expected by these musicians.

In short, bandwidth is vitally important in CMN because music is a performing art and because the notation is used in real time during performances. (Another interesting consequence of these facts about music is the importance of properly chosen page turns: see Sec. 2.3.6.4.)

Although CMN originated in a manner similar to natural language notations, it is quite different in that it is not based on any fixed character set, no matter how large, and in that it relies much more heavily on symbol position to convey information. We will discuss this difference in detail in Sec. 2.4, "A Comparison of Notation Systems".

It may appear at first that CMN is a gigantic "kludge" — a formidable hodgepodge of poorly organized and confused ideas which is simultaneously excessive in complexity and lacking in precision. Indeed, over the centuries dozens, if not hundreds, of attempts to reform music notation have been made, some superficial, some far-reaching.[3] I disagree strongly with the view that CMN is badly in need of reform. Some of CMN is indeed overly complex, and some of it is insufficiently precise. But for the style of music for which it evolved — if I can read purpose into the process in retrospect — CMN is a remarkably successful system. Furthermore, CMN has proven flexible enough to be adaptable in almost any direction. Rather than complain that a system that works superbly for Beethoven does not work so well for Xenakis, we should marvel that it works for Xenakis at all.[4],[5]

2.2. GENERAL DEFINITIONS

2.2.1. What Is Notation?

A standard unabridged dictionary, [MERR34], gives six definitions of "notation". The first three are not relevant to us. Definition 4 is

> act, process, or method of representing by a system or set of marks, signs, figures, or characters; any system of characters, symbols, or abbreviated expressions used in an art or science, to express technical facts, quantities, etc.

Definition 5 is of mathematical notation, and 6 (by far the longest) of music notation. More formally, Goodman [GOOD76] discusses notation from a logician's standpoint. It is beyond the scope of this dissertation to go into detail, but he distinguishes between notational *schemes*, the criteria for which are purely syntactic, and notational *systems*, which must satisfy semantic requirements as well. He concludes (p. 183):

> The main corpus of purely musical characters of the system [of conventional music notation] . . . appears on the whole to meet the semantic as well as the syntactic requirements for a notation. The same cannot be said for all the numerical and alphabetical characters that also occur in scores.

In any case, Goodman admits that his definitions are far from ordinary usage, and for our purposes the informality of the dictionary definition will be quite satisfactory. Any references in this dissertation to notational systems should be taken informally unless specified otherwise.

2.2.2. What Is Conventional Music Notation?

I have not really defined conventional music notation (CMN) yet. For the purposes of this dissertation, an informal definition will do: CMN includes *any arrangement of the symbols in general use by composers in the European art music tradition from about 1700 to 1935, used with the meanings that were standard: (1) if the notation was made between 1700 and 1935, at the time it was made; (2) if the notation was made before 1700, with the meanings of 1700; or (3) if the notation was made after 1935, with the meanings of 1935.* Thus, I am referring to a large set of conventions for semantics as well as syntax, although my primary concern is with syntax. The endpoints 1700 and 1935 are somewhat arbitrary.[6] As the above definition implies, and as I said in Sec. 2.1, music notation has actually changed continuously for centuries, although relatively slowly over the 1700 – 1935 period. Semantic changes within the period were more substantial than syntactic ones, but still not very great.[7] Note also that I do not insist that, to qualify, music must actually have been composed or notated between 1700 and 1935: most of the works of Elliott Carter, for example, are expressed in CMN, even though they were not written until the 1940's and after.

2.3. CONVENTIONAL MUSIC NOTATION FOR THE COMPUTER SCIENTIST

This section is devoted to a series of explanations and tutorial examples that assume no previous knowledge of music notation. I hope thereby to make the rest of the dissertation understandable to the nonmusician, especially the computer scientist. My intent in this section, then, is to impart a "reading knowledge" of CMN. To that end, I will concentrate on the *principles* of CMN; I will also discuss many, but nowhere near all, of the *details* of CMN. For example, the question of when note stems should go up and when they should go down is not discussed here. I will, however, point out a few "writing knowledge" problems to give a feel for the complexity of CMN. Two good introductions to CMN for the layperson are [SEEG73] and [SHAN57]. For more details from a musician's perspective, see the standard texts: [READ69], [ROSS70], and [STON80]. Three more specialized books, [DONA63], [GOMB75], and [READ78], are also useful. Finally, [RAST82] is a good historical study of Western music notation. Of all these, [ROSS70] is the only one that gives any significant number of explicit "writing knowledge" details, although it still omits far more than it includes. What [ROSS70] describes is also the closest to CMN; [READ69] and [STON80] include a great deal of new music notation that is not standard — not yet, at any rate. Chapter 4, "How SMUT Works", gives more explicit information than any of these sources on certain topics, for example rhythm notation and horizontal positioning of symbols (often called "justification", or, more accurately, "punctuation").

2.3.1. Definitions

Before going on, we need some additional terminology. Much more will be introduced as we go.

As I have said, most Western music consists of several synchronized lines going on simultaneously; these lines are technically called *voices*, regardless of whether they are performed by singers or instruments. Note that a multivoice composition may be played by one person, e.g., on the piano. The concept of "voice" is subtle and really needs more attention; we will return to it in Sec. 2.3.6.

In music in any of the styles that are notated with CMN, each voice consists of a series of tones, usually having either constant or (for percussion instruments) indefinite pitch, and intervals of silence. The symbols for the tones are called *notes*; for the intervals of silence, *rests*.[8]

A *score* is the music notation for a composition to played simultaneously by one or more performers giving the music to be played by all performers, with vertical alignment indicating the temporal relationships among notes and rests in different voices. If there is more than one performer, the performers do not ordinarily play from the score; instead each plays (or sings) from a *part*, showing only what she or he is to do.[9] In such a case, the score is for the use of conductors and persons studying the music.

A *system* is a set of two or more staves (see Sec. 2.3.2 for the definition of "staves") connected by a vertical line or brace at the left, describing the music to be performed simultaneously by one or more performers. In a score, every page is made up of one or more systems. Parts can also include systems, since some instruments, for example the harp and the piano, are written on more than one staff.

A *movement* is either a complete musical composition or the largest subdivision of a musical composition. Movements are nearly always separated from surrounding movements by major changes in the characteristics of the music and are usually separated by silence as well.

2.3.2. Pitch Notation

Fig. 1 shows the piano keyboard (in relation to two staves with clefs, which we will discuss shortly). It is well known that musical pitches are described by the letters A through G. These letters correspond to the white keys on the keyboard in a pitch range of one *octave*, i.e., a *frequency* range of 2:1 (under ordinary

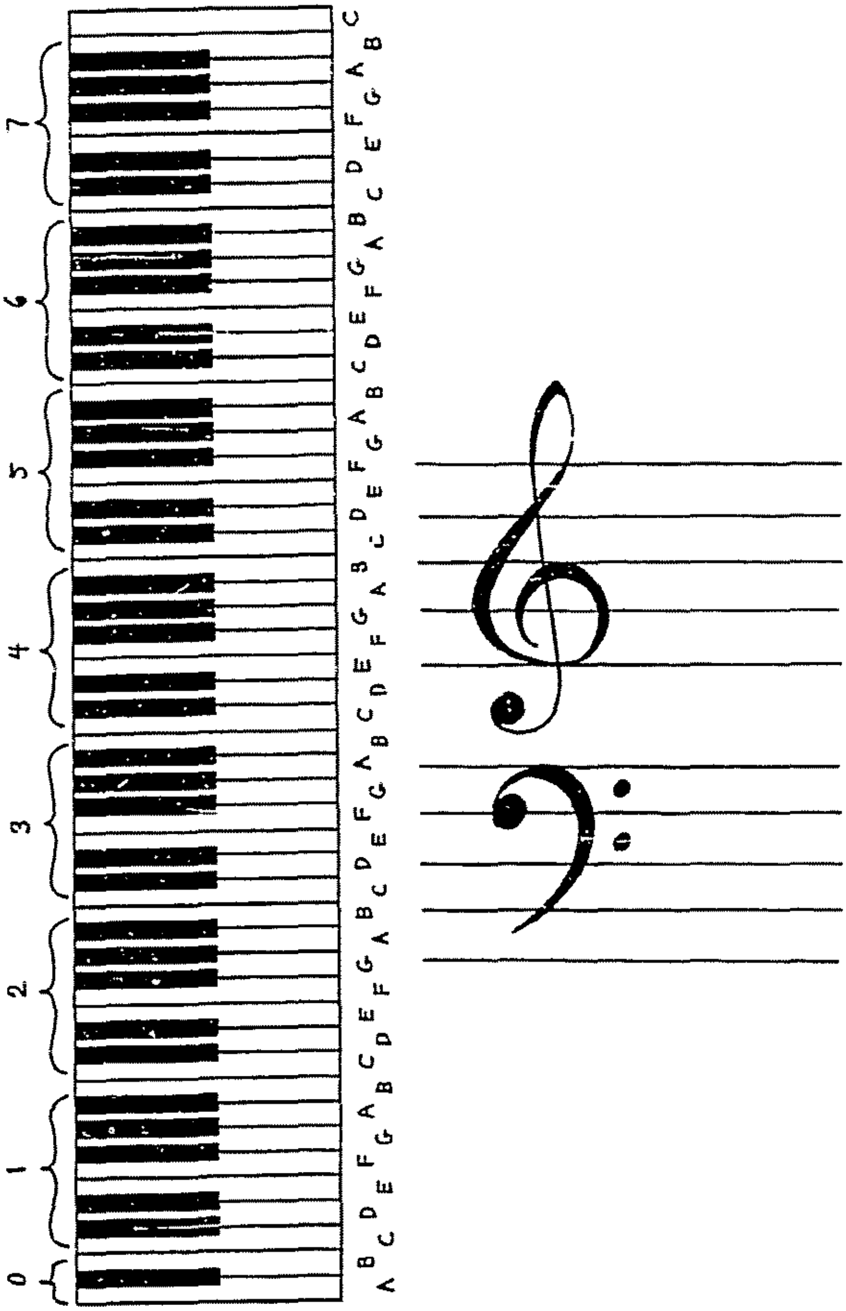

0
1
2
3
4
5
6
7
A B C D E F G A B C D E F G A B C D E F G A B C D E F G A B C D E F G A B C D E F G A B C D E F G A B C D E F G A B C

Figure 1. Piano keyboard and clefs

circumstances, pitch is logarithmically related to frequency); then they repeat.[10] Thus, the piano keyboard covers slightly over seven octaves. It has eight notes called "C", one of which is called "middle C" (circled in the figure). The eight C's are in the frequency ratio (from left to right on the keyboard) 1:2:4:8:16:32:64:128 and have actual frequencies from about 32.7 to 4186 Hertz (cycles per second). There are many systems for combining letter names with octave designations; by far the most sensible is that used by the Acoustical Society of America. In this system the octave of the keyboard starting with the lowest C is labelled 1. Octaves above (to the right) are labelled with increasing integers, octaves below (to the left) are labelled with decreasing integers — though numbers below 0 are hardly ever needed! In this system, then, middle C is "C4". So far we have labelled all of the white keys of the keyboard, but none of the black keys. We'll fill this gap shortly.

Fig. 2 shows a musical *staff* (plural *staves*), consisting of five parallel lines; this particular staff has a *treble clef* at its left end. As I have said, vertical position in CMN indicates pitch; however, the correspondence is not fixed, but depends on the clef. Notes are restricted to vertical positions centered on staff lines or on the spaces in between; each of these positions corresponds to the letter name of a different pitch. The letter names for treble clef are given in the figure. Note that E and F both occur twice, but in two different octaves, namely octaves 4 and 5. The other clefs are shown in Fig. 3. With *bass clef* instead of treble, the bottom line of the staff would be G2 instead of E4. The correspondence between position and pitch for treble and bass clefs is shown in Fig. 1. The less common *alto* and *tenor* clefs respectively make the bottom line of the staff F3 and D3. Rigid conventions determine which clef or clefs can be used for a given instrument or voice. Only treble and bass clefs are ever used for the piano, for example.[11]

In Fig. 4 a note is added to the staff, specifically a whole note (this term is defined in Section 2.3.3.1) with pitch E5. In Fig. 5 a *flat* is added, which lowers the pitch to *E-flat 5*. Now we are ready for the black keys. Fig. 6 again shows the piano keyboard, this time only one octave but with all keys labelled. Clearly, counting both black and white keys, there are 12 notes in an octave; the 13th note repeats the initial one an octave higher. These are equally spaced on a

Figure 2. Staff and treble clef

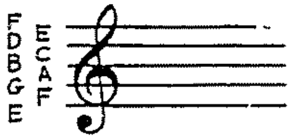


Figure 3. Other clefs

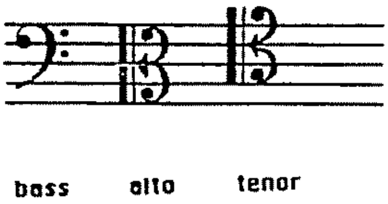


Figure 4. Whole note

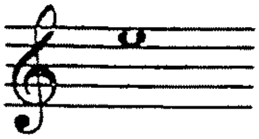
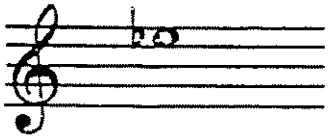


Figure 5. Flat



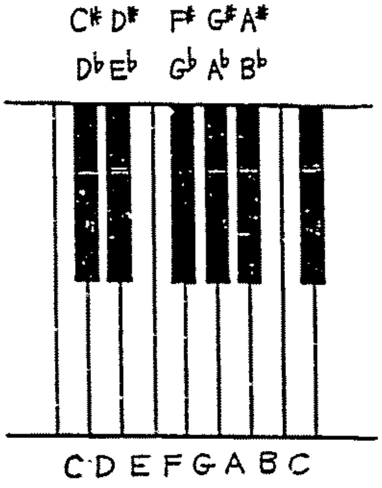


Figure 6. Black and white keys

logarithmic scale of pitch [12], and the distance between any two consecutive notes is called a *semitone*. However — to oversimplify considerably — at critical points in the history of Western art music, only seven of the 12 were used at a time to make up the basic pitch material of a composition, with the other five being used for embellishment; many aspects of pitch notation reflect this historical inequality of importance. One such reflection is, of course, the arrangement of the piano keyboard into black and white keys, and others are the facts that only the white keys have independent names (the letters A through G) and positions on the staff. The five black key pitches may be obtained by *accidentals* -- semi-local pitch modifiers — two of which are the *flat* (down one semitone) and the *sharp* (up one semitone). (Fig. 7 shows the five accidentals: double flat, flat, natural, sharp, and double sharp. Double flat and double sharp have the effect one would guess, namely two semitones in the appropriate direction each; they are used only under special circumstances. I will discuss the natural shortly.) Since there is no black key between B and C, B-sharp is the same as C, and C-flat is the same as B.[13] Likewise, E-sharp is the same as F, and F-flat is the same as E. I call accidentals "semi-local" because they affect, not just the following note, but all notes with the same letter name and octave until the end of the measure (defined in Sec. 2.3.3.2).[14]

In Fig. 8 the flat accidental has been removed and a *key signature* containing two flats has been added. The key signature is a *global* pitch modifier, global in two senses. One is its scope: it applies, not merely for the rest of the measure, but for the rest of the movement or until explicitly changed. The other is vertical applicability: if B-flat and E-flat appear in the key signature (as here), B's and E's in *all* octaves are flatted, not just in the octaves where the flat signs appear. How, then, is it possible to write unflatted E if E-flat appears in the key signature? By putting another accidental, a *natural*, in front of the note (Fig. 9). To restate our accidental-scope rule, any explicit accidental — double flat, flat, natural, sharp, or double sharp — overrides the key signature or previous accidentals, with a scope of the rest of the measure.

We now know enough notation to be able to notate most pitches in the middle of the keyboard, namely those that happen to fall on the staff in whatever clefs the instrument allows — for the piano, treble and bass. Unfortunately, most of

Figure 7. Accidentals

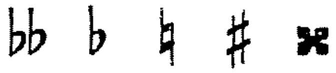


Figure 8. Key signature

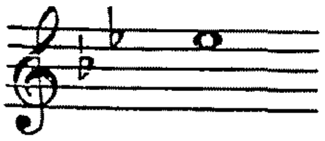


Figure 9. Natural

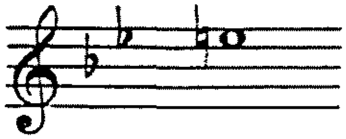


Figure 10. Ledger lines

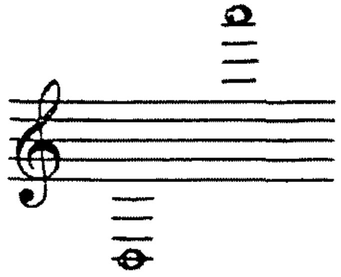
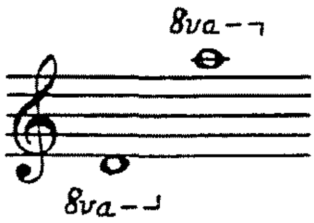


Figure 11. Octave signs



the keyboard does not satisfy this condition. We can't even notate middle C (C4), which (as Fig. 1 shows) is above the staff in bass clef and below it in treble clef! Two devices exist to solve this problem: *ledger lines* and *octave signs*. Ledger lines (Fig. 10) are simply temporary extensions of the staff upward or downward. A very convenient way to extend the staff's pitch range by another octave and a half is with four or five ledger lines (more or less). With more than this, it is too easy to misread the number of ledger lines, and something else has to be done. That something is the octave sign (Fig. 11), which temporarily moves all pitches up or down an octave: up if the octave sign is above the staff, down if it is below.[15] The notes in Fig. 11 have the same pitches as those in Fig. 10, namely D3 and A6. As the figure shows, ledger lines and octave signs can be used together.

2.3.3. Time Notation

2.3.3.1. Notes and Rests Alone: Duration

As I have said, time is represented in CMN on the horizontal axis: the music is read from left to right (except, of course, across line breaks).

Of all the symbols in music notation — clefs, key signatures, notes, rests, barlines, etc. — the only ones that actually occupy time in performance are notes and rests. In a given musical voice, notes and rests are performed "end-to-end", each one beginning at the moment the previous one ends. The *duration* of a note or rest is just the amount of time it occupies. (The term *rhythm* refers to much more complex and higher-level concept; we will postpone discussing it for a while.) The common basic durational symbols for both notes and rests are shown in Fig. 12*a*, and some of the relationships among them are shown in Fig. 12*b*. A sixteenth-note is shown with its parts labelled in Fig. 13.[16] It can be seen that the names of the durations are mostly ordinals of powers of 2. The basis of time notation is the idea that the duration of one specific symbol is specified (sometimes *very* implicitly, not to mention vaguely); all other symbol durations are relative to that one, in an extended binary system. For example, a piece might be marked "M. M. ♩ = 60", an explicit statement that there are to be 60 quarter-notes per minute. (This marking is called a *tempo mark*. Tempo marks set a global variable relating to time; they

Figure 12*a*. Symbols for note and rest durations

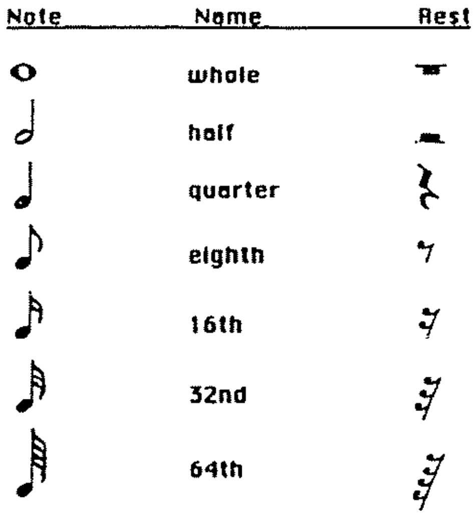


Figure 12*b*. Duration relationships

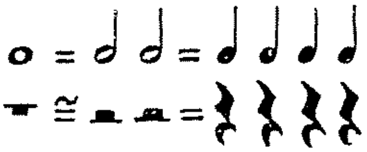


Figure 13. Parts of a 16th-note

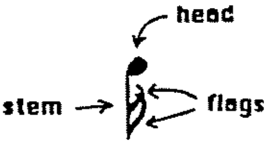


Figure 14. Tie

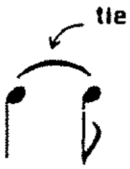
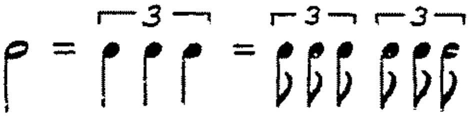


Figure 15. Triplet



are simply positioned horizontally at the point where they are to begin taking effect. Other tempo marks — "accelerando", "ritardando", etc. — describe gradual changes of tempo, so that, on a graph of tempo against time, more-or-less general line segment functions are available.) Naturally, quarter-notes last half as long as half-notes and twice as long as eighth-notes, four times as long as 16th-notes, etc. There is one exception to this scheme: the whole *rest* has the duration of a whole *measure*, which may or may not equal a whole-note or two half-rests. (See below for the definition of measure.)

Clearly the list of durations that can be notated in this way has intolerably large gaps. What if "third-notes" are wanted? These gaps are filled in in several ways.

(1) The most general extension is the *tie*, a curved line that combines two notes of the same pitch into one continuously-sounding note (Fig. 14). Ties can be used consecutively, so that three or more notes are combined into one.[17]

Ties are very general, but alone are not sufficient to satisfy the needs of real music. There are two specific situations that are still not adequately taken care of; each has a corresponding notational device.

(2) The more obvious situation is the occurrence of durations whose ratios to any basic duration cannot be expressed in binary with any finite number of digits, and which would therefore require an infinite number of tied notes. The "third-note" is an example. This is handled by a device that, unfortunately, has at least six names. We will call it the *groupet*; it is also known as the *artificial, irregular,* or *extrametric group*, and (less sensibly) as the *unequal rhythmic group* or *irrational rhythmic subdivision.* By far the most common example is the *triplet* (Fig. 15), whose three notes[18] are each reduced to 2/3 of their normal durations. (The little "3" is called an "accessory numeral"[19].) Unfortunately, as in so many aspects of CMN, as soon as one leaves the most common situations, the rules become both complex and vague. The question here is what durational values should be used within the groupet. Consider, for example, two ways of notating a septuplet with a total duration of a half-note (Fig. 16). The first way follows the triplet example by using within the

Figure 16. Two ways of notating a septuplet

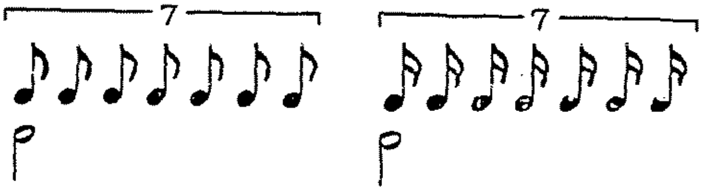


Figure 17. Augmentation dot

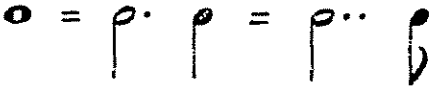


Figure 18. Grace notes

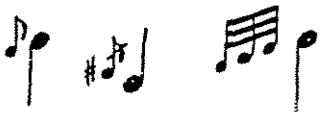
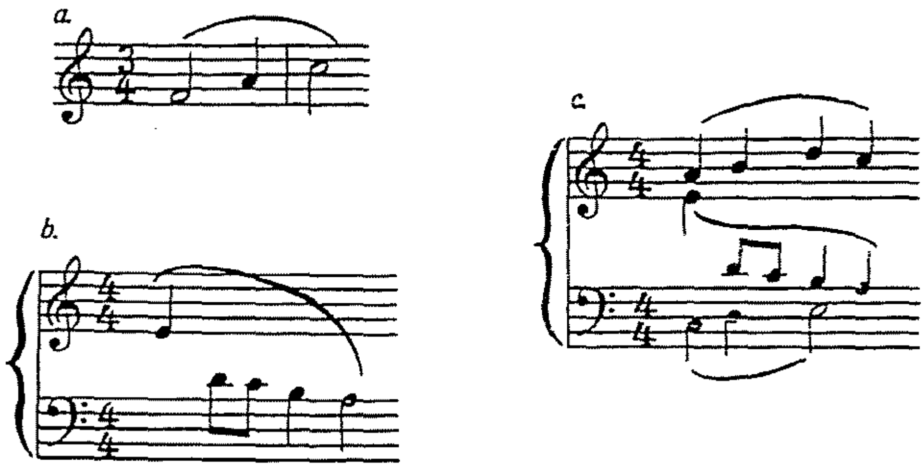


Figure 19. Slurs



groupet eighth-notes whose durations are greatly reduced: 7/8 >> 1/2. (The rule here is to "retain the note-values of a regular note-division for the faster, irregular pattern, until the next, shorter regular division, and thus the next shorter note-value has been reached." [STON80, p. 130] Under this rule, groupets always reduce durations.) The second way uses within the groupet sixteenth-notes whose durations are slightly increased: 7/16 < 1/2. The (relatively few) proponents of the second method argue that the inconsistency of having groupets sometimes shrink durations and sometimes stretch them is more than compensated for by the advantage of using notes whose effective durations are much closer to the normal ones. The situation is further complicated by some musicians' using one method in binary rhythmic division levels (see Sec. 2.3.3.2) and the other in ternary division levels. All one can say in general is that notes and rests within groupets always have more than half their normal durations, and *nearly* always less than their full normal durations. Fortunately, it is almost always possible to tell from context what the composer intended.[20] Nested groupets are also possible, but they are relatively rare.

(3) The other problem situation is purely a matter of readability: notes with durations of one-and-a-half times a basic duration are extremely common, and without some additional notational machinery, music would be crowded with thousands of ties. The *augmentation dot* (Fig. 17) allows a single note to represent not only these durations, but a superset of them, namely those of the form $\sum_{k=0}^{n} d \times 2^{-k}$, where d is one the basic durations. Specifically, the above formula describes a note with n augmentation dots; $n=0$ or 1 is common, 2 fairly rare, 3 or 4 very rare.[21] The vertical position of the dot depends on whether its note is centered on a space or a line. Dots belonging to "space" notes always appear directly to the right of the notehead; dots belonging to "line" notes appear in the middle of the next space up if there is only one voice on the staff. If the staff is shared with other voices, dots belonging to "line" notes may appear in either the space just above or the one below. For more on this question, see Sec. 2.3.6.2.1.

CMN also has a facility for describing notes of indefinite but (usually) short duration, informally called *grace notes*. Their notation is just like that of ordinary notes, except that they are smaller and often have slashes through their stems (Fig. 18). No additional time is allocated for grace notes; they usually take time from the preceding, but sometimes from the following, note or rest.

2.3.3.1.1. Articulation and Phrase Marks. The actual durations for which notes are sounded are usually less than the written durations, the missing time being filled with silence. The fraction of the total written duration that is sounded is affected by symbols called *phrase marks* and *articulation marks*. The *slur* is the standard phrase mark. Graphically, slurs (Fig. 19) are curved lines, in high-quality notation thickened in the middle; they extend from one note to a later one, frequently many notes later. Figs. 19*a* and *b* show slurs with zero inflection points, *c* a slur with one inflection point; the latter two figures involve two staves, as in piano music. It is quite difficult to describe the set of possible shapes of slurs precisely, but I will say more about them in Sec. 2.4. As Fig. 19 shows, slurs can look very much like ties, and, in fact, the possible shapes of slurs are a superset of those of ties. Their meanings are also related to those of ties: each note in the compass of a slur should occupy its full written duration, with silent time reduced to zero. Henceforth in this dissertation "slur" should be taken to mean "phrase mark, slur or tie" unless otherwise noted.

Other articulation marks apply to a single note at a time and can, among other things, either decrease or increase the proportion of the note's time that is sounded. For examples of articulation marks, see Fig. 33c.

2.3.3.2. Notes and Rests in Context: Rhythm

The chapter on "Duration and Rhythm" in [STON80] begins with the statement (p. 81):

> In all other chapters, the notational signs and procedures are presented as isolated phenomena, divorced, with very few exceptions, from any musical context. This cannot be done with rhythm. For rhythm to be perceptible as such, a context is essential. Stated more simply: a sixteenth note by itself is

> no more than a time value; it becomes a rhythmic phenomenon only in the context of what happens immediately before and after.

This context dependency makes rhythm one of the most difficult aspects of music to discuss or even to define. For our purposes, however, we may consider any particular rhythm to be a periodic pattern much like a "ruler function" that describes the relative perceptual salience of notes and rests. Ruler functions are so called because of the similarity in appearance of a graph of one to a series of tick marks on a ruler; see Fig. 20.[22] The horizontal axis in the graph represents time, the vertical axis salience. I will use the term *rhythmic strength* to refer to the relative perceptual salience of a given temporal point in a specified meter.[23] The different rhythmic strengths can well be considered to establish a hierarchy of levels. Rhythm is expressed in sound in several more or less subtle ways, the most important of which is by increasing slightly the loudness of notes that begin at rhythmically strong moments. It is expressed in CMN primarily by such symbols as time signatures and barlines and secondarily by modifications to notes such as substituting beams for flags. Almost all music written in CMN is divided into *measures* (also called *bars*) which are delimited by *barlines* (Fig. 21), a measure being everything between two barlines. A *time signature* or *meter signature* (Fig. 22) is a symbol, most often one integer above another[24], that specifies the *meter* of the music and thereby indicates two things: the total (relative) duration of, and the underlying rhythmic structure of, each measure.[25] Unfortunately, the two things are too complex to be encoded in a straightforward way in the two integers. For example, 3 above 4 (which we will write "3/4") means the duration of the measure is three quarter-notes and the rhythmic structure is that given in Fig. 23; 4/4 means the measure duration is four quarter-notes (one whole-note), and the rhythmic structure is that of Fig. 24. 6/8 describes a meter with the same measure duration as 3/4, but with a different rhythmic structure, namely that in Fig. 25. These examples suggest another way in which rhythm is like tick marks on a ruler, namely that divisions on all levels are usually binary, but need not be. Nonbinary divisions occur in music only at the top level or two — rarely, at the top three levels — and when they do, according to most experts, they are always ternary.

Figure 20. Rhythm

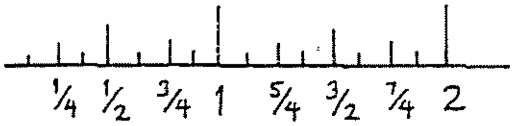


Figure 21. Measure

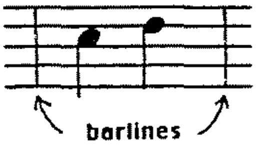


Figure 22. Time signature

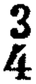


Figure 23. 3/4 meter

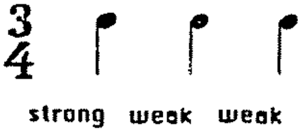


Figure 24. 4/4 meter

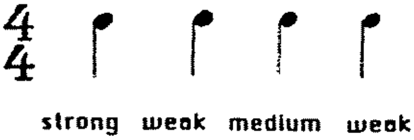


Figure 25. 6/8 meter

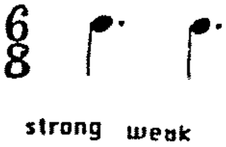
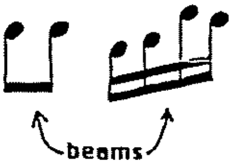


Figure 26. Beams



2.3.3.2.1. The Classification of Meters. Several terms are in use to describe the different types of meters; in order to discuss SMUT's rhythm clarification and automatic beaming (see Sec. 2.3.3.2.2) capabilities later on (Sec. 4.4), we will need to know what they are. First, though, we need another piece of terminology. A *beat* is one of a series of temporal points that, in the words of an elementary text, [WINO79], recur regularly for any given section of music

> at a rate of speed that is appropriate or "natural" for the music. Sing a march or a college "fight song" and clap as you sing. You will probably find that you are clapping at a rate of about 120 to 140 claps per minute. You are clapping the beat.

Another definition of "beat" is given in a standard music dictionary [APEL69]: "the temporal unit of a composition, as indicated by the up and down movements, real or imagined, of a conductor's hand."

[APEL69] says about meters (emphasis mine):

> According to whether there are two, three, or four units [i.e., beats] to the measure, one speaks of *duple* (2/2, 2/4, 2/8), *triple* (3/2, 3/4, 3/8), or *quadruple* (4/2, 4/4, 4/8) meter... All these are *simple* meters... *Compound* meters are simple meters multiplied by three: *compound duple* (6/2, 6/4, 6/8), *compound triple* (9/4, 9/8), *compound quadruple* (12/4, 12/8, 12/16).

This rather confusing terminology may be best illustrated by comparing several time signatures, each of which describes a measure with a total duration of three quarter-notes: 3/4, 6/8, and 12/16. See Table 1. The boldface indicates the beat in each case. Taking 6/8 as an example, the table shows that a measure is composed of two dotted quarter-notes, each of which is a beat: this is the top level. Each dotted quarter is composed of three eighths, which is the second level. Each eighth, in turn, is composed of two sixteenths, which is the third and last level shown in the table, though one could continue the process indefinitely. Note that across the three meters in the table the durational units on the top two levels vary, but in each case the third level consists of 16th-notes.

[APEL69] does not mention meters with more than four beats, but they certainly exist; for example, 5/4, 15/16, and 7/8 are sometimes called

TABLE 1. Time Signatures and Their Usual Structures

Division level	3/4	6/8	12/16
Top	**3 quarters**	**2 dotted quarters**	2 dotted quarters
2nd	2 eighths	3 eighths	**2 dotted eighths**
3rd	2 16ths	2 16ths	3 16ths

"Division levels" are of the measure. Entries in bold indicate beats.

mixed meters, since the beats are grouped alternately in twos and threes.

Now we are ready to consider the simple-compound dichotomy. It relates to what happens on the first level *below* the beat: meters are simple if the top-level division of the beat is into two parts, and compound if the top-level division of the beat is into three parts. Thus, 3/4 is simple triple; 6/8 is compound duple; and 12/16 is compound quadruple. A good rule of thumb, used by my program SMUT, is that if the "numerator" is a multiple of 3 but greater than 3, the meter is compound and three of the "denominator" durations form a beat. Otherwise the meter is simple and one of the "denominator" durations is a beat. Note that this is only a rule of thumb, not a decision procedure. None of the standard texts gives any rules whatsoever, but the examples all give are compatible with this rule.

2.3.3.2.2. Beams. Flags on notes are very often replaced by *beams* (Fig. 26). The basic function of beams is to increase the readability of music by grouping notes according to the underlying rhythm. In Read's words [READ69, p. 81]:

> No comment is necessary to point out the superior clarity of the beamed example below as opposed to the contrasting dense forest of individual flags outlining the same pattern (Fig. 27). The visual advantages of beaming can be further illustrated by a comparison of the following two versions of a compound two-beat meter . . . (Fig. 28)

Figure 27. Beams vs. flags

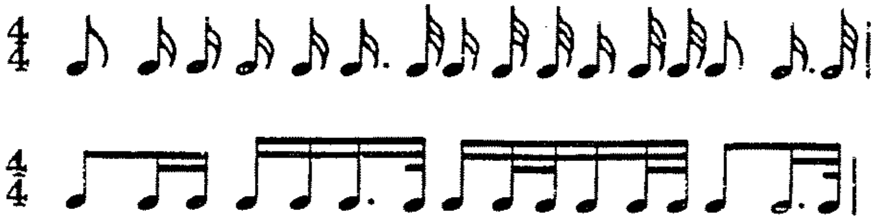


Figure 28. Beams vs. nothing

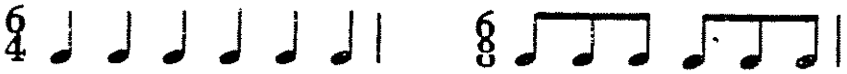
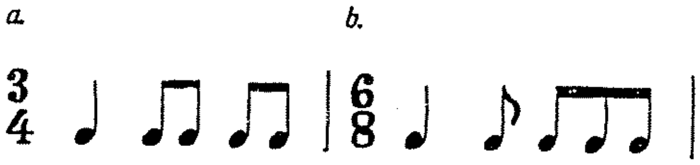


Figure 29. Beams and the underlying rhythm



6/4 time is like 6/8 in consisting of two groups of three units each, not three groups of two units: both meters are compound duple. But in 6/8 the units can be beamed; in 6/4, they cannot. One more example: the 3/4 time passage in Fig. 29*a*, if written in 6/8 time, would probably be beamed as in Fig. 29*b*. However, as usual, there are no hard and fast rules; in particular, the rhythmic level included by beams is indefinite, but is influenced by rules like "don't include too many notes in a beamed group."[26] Very roughly, "too many" probably means any number larger than about 8. In Figure 29*b*, we beamed together half the measure — three eighths — but with 32nd notes in 6/8 time, that structure would produce 12 notes in a group, which would be hard to read (Fig. 30*a*). One solution would be to beam together only four-note groups (Fig. 30*b*). The best solution might be to "structure" the beams, as in Fig. 30c. This piece of notation illustrates a distinction that is ocasionally useful, between *primary beams* (which include the entire beamed group) and *secondary beams* (which do not). In Fig. 30*c*, there is only one primary beam (the long one on the bottom).

There are several more peculiarities of beams; the only one I'll illustrate is the so-called *fractional beam*, where a beamed group includes an isolated note that requires more beams than either of the adjacent notes (Fig. 31). The question of which way the fractional beam should point is complex. In Fig. 31*a*, there is no ambiguity, since it must stay within the beamed group. In Fig. 31*b*, the fractional beams are pointed so that the rhythm is clearly shown.[27]

2.3.3.2.3. Rhythm Concluded. Movements of real music frequently begin with partial measures, informally called "upbeats" and formally termed *anacruses* (the singular is "anacrusis").[28]

For a full discussion of some aspects of proper rhythm notation, developing the rhythm clarification and fractional beam pointing algorithms used by SMUT, see Sec. 4.4.

Figure 30. Beam structure

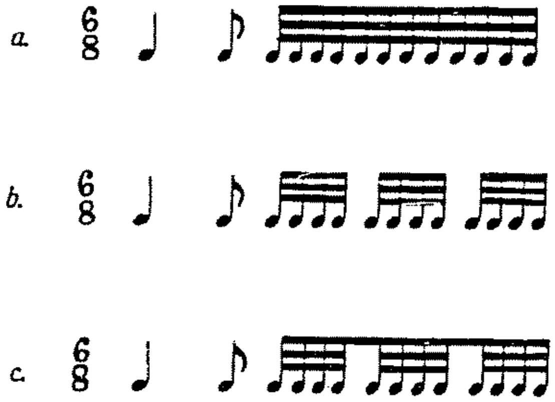


Figure 31. Fractional beams

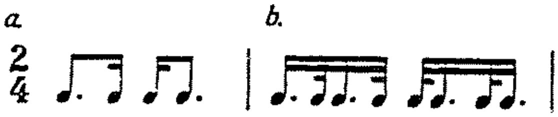
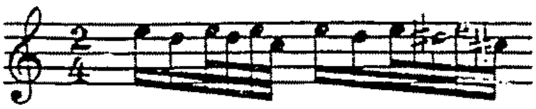


Figure 32. Horizontal spacing



2.3.3.3. The Horizontal Axis

As I have said, time in CMN is represented fairly explicitly along the horizontal axis. As one might infer from the presence of evenly spaced horizontal reference lines — the staff lines — but not vertical ones, the rules here are far less definite than for the representation of pitch on the vertical axis. The basic idea is that horizontal distances *suggest* time intervals but do not specify them: there is no isomorphism. Specifically, the ideal distance between consecutive time-occupying symbols — notes or rests — increases with the amount of time separating them, but more slowly than linearly. It would be too wasteful of space, in a piece that includes 64th notes, to allocate whole notes 64 times more room. We will discuss specific rules in Chapter 4, "How SMUT Works". In any case, several factors result in the continual violation of ideal spacing rules. For example, the presence of accidentals and dots on short notes may force them to be given more room than they would otherwise get (Fig. 32). The existence in music of multiple voices which must be synchronized profoundly affects horizontal positioning rules; see Sec. 2.3.6.2, "Complications Involving Multiple Voices", below.

2.3.3.4. A Non-CMN Approach to Time: Spatial Notation

A very different and generally much simpler approach to notation of time has achieved fairly wide use in 20th-century music. *Spatial* or *proportional* notation is definitely not CMN; however, it is worth briefly describing here, since it is of some relevance to computer music setting (see, for example, [MAXW83]). Spatial notation differs from CMN only in notation of time, in which respect it is somewhat like player-piano rolls: there are no time signatures and no quarter-notes, eighth-notes, etc.; instead, the horizontal axis is linear, and time is proportional to distance. Noteheads are not connected to flags or beams of the ordinary kind, but (unless very short) are followed by horizontal lines called "extenders" to show how long they are to be held. Thus, the time component of spatial notation is clearly not (in the terms of Goodman; see Sec. 2.2.1) a "notational scheme": precise interpretation of time values is not possible. For an extensive discussion of spatial notation, see [STON80].

2.3.4. Notation of Loudness

The notation of loudness in CMN is not only relatively nonexplicit, it is also relatively simple and undeveloped. The most explicit it gets is the appearance of verbal comments — usually in Italian and frequently abbreviated — plus a few special symbols, just above or below the notes they affect. Most of the common markings with their abbreviations are given in Table 2. (The "hairpins" are usually horizontal but need not be.) As is evident, most of the markings simply specify dynamic levels; a few, namely the hairpins, indicate gradual change from one dynamic level to another. The scope of markings of the latter type is sometimes indicated clearly, sometimes not. Thus, if we assume that "gradual changes" are linear, these markings make it possible to specify loudness as a line-segment function of time.

TABLE 2. Dynamic Markings

Symbol	Name	Meaning
ppp	pianississimo	as soft as possible
pp	pianissimo	very soft
p	piano	soft
mp	mezzopiano	moderately soft
mf	mezzoforte	moderately loud
f	forte	loud
ff	fortissimo	very loud
fff	fortississimo	as loud as possible
<	"hairpin"	increase loudness
cresc.	crescendo	increase loudness
>	"hairpin"	decrease loudness
dim.	diminuendo	decrease loudness
<>	"swell"	increase, then decrease loudness

2.3.5. Notation of Timbre

Notation of timbre, i.e., tone quality, in CMN is so nonexplicit that many works of 19th-century composers do not contain a single timbre indication other than the bare indication of which instrument (or voice) is to make which notes, and many works of earlier composers do not even include that. J. S. Bach's great *Art of the Fugue*, written in 1750, contains barely a hint as to what performance medium he intended. Furthermore, when timbre notation does appear, it is nearly always expressed by instructions that tell the performer what to do to produce the desired timbre — not with a description of the timbre itself.[29] For notational purposes, though, all that matters is that, like loudness, timbre is notated mostly with verbal comments above or below the relevant notes. For example, string instrument music frequently contains the indications "pizzicato" and "arco": respectively, "pluck the strings", and "use the bow" (the latter being the default). The scope of each of these markings extends until it is explicitly overridden. However, if a composer wants some notes plucked while others are bowed, a non-verbal symbol is available: the plucked notes can be indicated with a little "+" just above or below the noteheads.[30] There are also one or two types of timbre information that are conveyed by actually changing the shapes of the noteheads, e.g., to diamonds or "X"s.

Even though we supposedly have now covered the notation of all four parameters of musical sound, I have just mentioned for the second time (cf. Sec. 2.3.3.1.1) an important feature of music notation: small symbols associated with a note that appear just above or below the notehead. Discussion of this phenomenon really belongs in the next section.

2.3.6. Why Things Are Really More Complicated Than They Seem

There are several reasons why CMN is really more complicated than the presentation above suggests, even though the basic principles are as described. Some of these reasons apply to single-voice music and to individual voices in multi-voice music; some are a result of interactions between voices in multi-voice music. Still others are not on the level of voices at all, but relate to page layout.

I said in Sec. 1.3.1 that most of the really hard problems of music notation are exacerbated by complexity of *texture*. The term is difficult to define. It was adopted from its usage with textiles, where it refers to "the character of a woven fabric resulting from the arrangement, size, quality, etc., of the fabric's threads." [WORL64] Texture in music notation has to do with voices and staves and their interrelationships. Any example of single-voice music has the simplest possible texture. Slightly more complex is music on any number of staves with exactly one voice on each staff; more complex still is music on any number of staves with two voices on some staves; and the most complex textures — and those that are most difficult to notate — involve staves with three or more voices and/or voices crossing staves, i.e., not belonging always to the same staff.

Having said this, I must add that the term *voice* is ambiguous. When, for example, a pianist plays a melody accompanied by a succession of three-note chords, the chords might be counted as three voices or as one voice that is "thickened". The latter sense is more useful from the standpoint of notation, and it is usually — though not always — the sense I intend.

2.3.6.1. Complications within a Single Voice

Our "modified coordinate system" model of CMN is further modified by the fact that certain small symbols can appear above or below any notehead; these symbols can affect *any* of the parameters — pitch, time, loudness, and timbre — of that note alone. Common examples in each category of these symbols, which we will call "note modifiers", are given in Fig. 33.[31]

(a) Pitch modification: microtonal (less than a semitone) pitch modifiers. These are really not standard CMN, however.

(b) Time modification: *fermata* or *hold*, *staccato mark*, *tenuto mark*. These last two are articulation marks, which we have mentioned before (in Sec. 2.3.3.1.1).

(c) Loudness modification: *accents*.

(d) Timbre modification: "+" (plucked string, mentioned above), "0" (open string), etc.

A related but more complex phenomenon is that of *ornaments*, pitch modifiers whose notation is usually similar to that of the simpler note modifiers. The main difference is that ornaments are like macros, each one expanding (often in a very ill-defined manner) into a series of shorter notes

Figure 33. Note modifiers

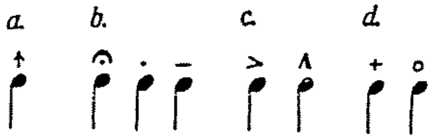


Figure 34. Ornaments

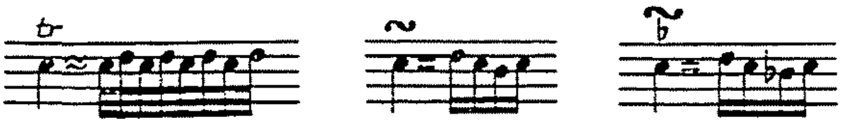
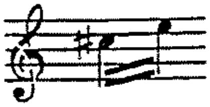


Figure 35. Fingered tremolo



whose total duration is equal to that of the original note. See Fig. 34 for several common examples, namely one *trill* and two *turns*. The trill always involves rapid alternation between the given note and one a step higher, but the starting note, the speed of the alternation, and the total number of notes are undefined. Ornaments are like macros even to the extent of allowing parameters; the second turn in the figure illustrates this. An ornament that is notated quite differently is the *fingered tremolo* (Fig. 35); it is actually a generalization of the trill with an arbitrary distance between the two pitches.

Finally, character strings are used in CMN for many purposes. Three that have already been discussed are tempo marks, indications of loudness, and instructions related to timbre. Two more usages, both important, are:

(1) In vocal music, to give the "lyrics", the words to be sung. This is done simply by placing them below the appropriate staff, the first letter of each word or syllable aligned with the note it is sung to, or, if the word or syllable is sung to several notes, with the note on which it begins.

(2) In any kind of music, for miscellaneous verbal directions to the performers. These are most often in Italian. Many are indications of the general character or mood of a composition or passage, for example *"dolce"* (sweetly), *"leggiero"* (light or nimble), *"tout devient charme et douceur"* (all becomes charm and sweetness), etc. Other directions to performers are harder to classify. In keyboard music one finds markings like "L.H." (play with the left hand) and so on.

The various usages of character strings are differentiated partly by positioning and partly by different typefaces.

2.3.8.2. Complications Involving Multiple Voices

All the complications within a single voice just discussed are relatively easy both to describe and to deal with. Those involving multiple voices are neither. Multiple-voiced music can be notated in either or both of two ways: with multiple staves, and/or with multiple voices on a single staff. Multiple staves bring in some additional complexity, but the overwhelming majority of difficult situations arise with multiple voices on a single staff.

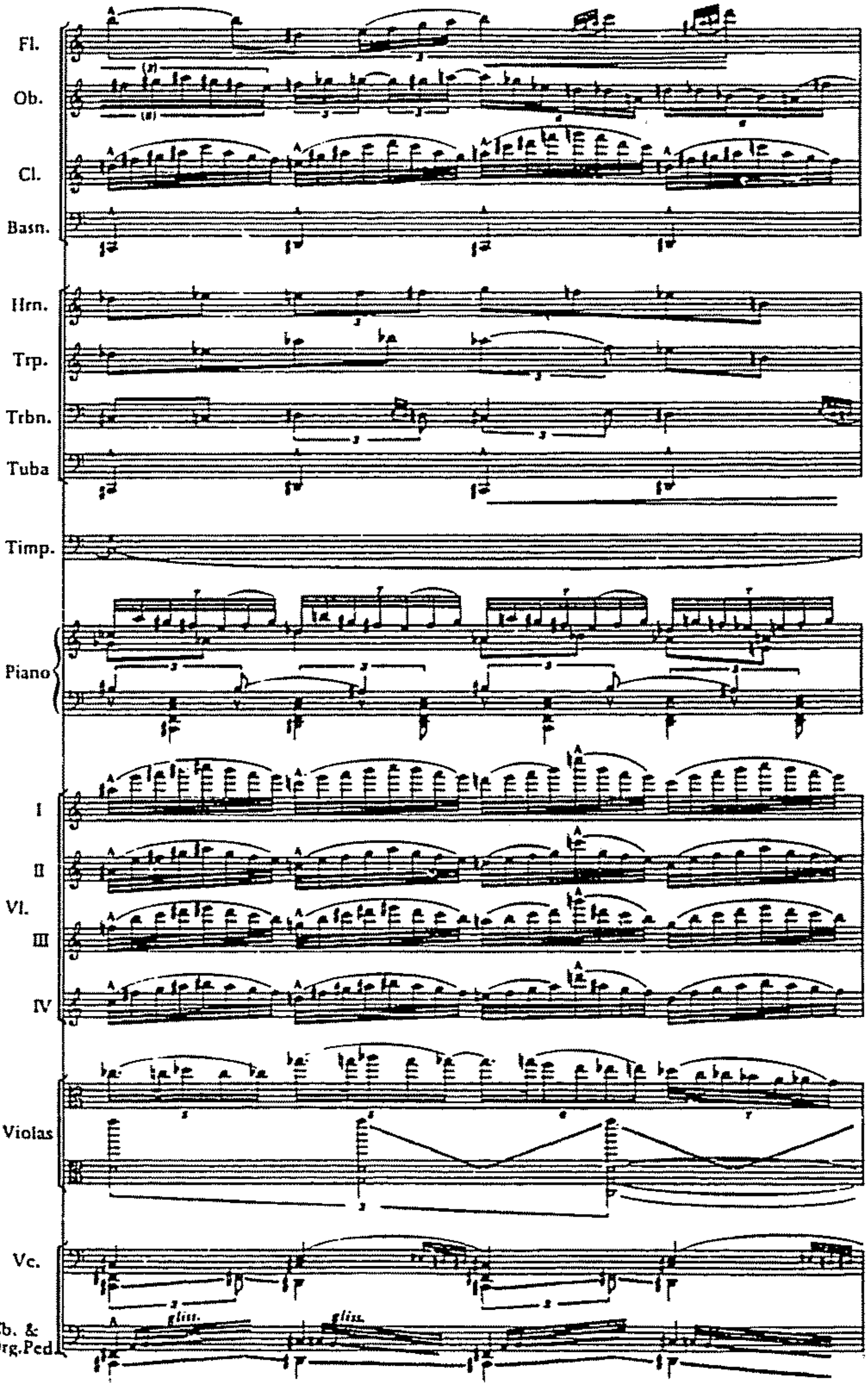
2.3.6.2.1. Two or More Voices Sharing a Staff. Two voices sharing a staff are basically accommodated by some simple modifications of the standard rules. For example, stems for the upper voice are always pointed up and for the lower voice down, and slurs, ornaments, and groupet accessory numerals all move to the outside. Many problems can arise, however. For example, the voices can cross, so that the upper voice is temporarily lower in pitch; in this case interference between symbols in the two voices, requiring some repositioning, is quite likely. With more than two voices on a staff, nothing is straightforward. (Bach's *Six-Part Ricercare* from *The Musical Offering* is a spectacular example: it has three voices per staff almost continuously.) In fact, *variable* numbers of voices on a staff and voices moving from one staff to another are not at all unusual; these occurrences sometimes leads to ambiguities when fewer than the maximum number of voices are present. [STON80] includes some excellent discussion of such problems.

One of the most difficult problem areas with multiple voices sharing a staff relates to horizontal positioning.

2.3.6.2.2. The Horizontal Axis Revisited. At this point, we need another piece of terminology. A *hunk* is a set of symbols in CMN that includes one or more notes, all beginning at the same time, and that, from a notational standpoint, behaves as a "nearly" inelastic unit.[32] That is, the entire hunk can be moved left or right, but no symbol in it can be moved independently more than a small distance. (Hunks cannot be moved vertically, since this would change the pitch of the note(s).) Although the notes and/or rests in a hunk must all begin at the same time, they may have different durations and so may end at different times. A complicated hunk might include several noteheads on a common stem, accidentals and dots for some, and an accent mark.

Horizontal positioning in multi-voice music is much more complex than in single-voice music. For one thing, since time is indicated on the horizontal axis, intuition demands that any note or rest beginning after another be indicated by its positioning further to the right, even if the notes are in different voices. In fact, in CMN, this is always done. Fig. 36 (page 85 of

Figure 36. Ives: *Three Places in New England*, "The Housatonic at Stockbridge".



an orchestral score, "The Housatonic at Stockbridge" from Charles Ives' *Three Places in New England* (1911; Mercury Music, 1935)) shows an exceptionally complex situation. Look at the rhythm starting one quarter-note into the measure and ending two-quarter notes in. Here six different instruments are to play two, three, five, six, seven and eight notes, respectively, in the same amount of time, and therefore the notation interleaves two, three, five, six, seven and eight hunks in the same amount of space. Thus, if the voices were superimposed, their hunks would interleave in the correct order.

Intuition also suggests strongly that synchronization be indicated by exact vertical alignment. Again, the rule follows intuition, but in this case significant exceptions can occur when multiple voices occur on a single staff. The rules depend on whether the voices have separate stems for their noteheads or share stems. Fig. 37 illustrates the former case, Fig. 38 the latter. In each figure, see *a* for the normal positioning of two simultaneous notes. *b*, *c*, and *d* illustrate the two basic problem situations, both involving notes close in pitch, in which standard practice is to move one notehead to the side. Positioning of the augmentation dots is also affected here. The first situation (*b*), where the notes are on adjacent space and line positions, is relatively straightforward and, unless the voices cross in two-stem notation, is always handled in the ways shown. The second situation (*c* and *d*), where the notes have exactly the same vertical position, is more complex: in two-stem notation, if the notes have the same type of head, same accidental, and same number of augmentation dots, and both have stems (i.e., they are half-notes or shorter), than a single head is used with stems going both up and down (Fig. 37*c*). If one or more of these conditions is violated, as in *d* and 38*c*, two heads side-by-side must be used. ([STON80, pp. 125 – 26], makes the potential difficulties here quite clear.) The presence of accidentals in both voices can make matters still worse, as in *e*, an "altered unison"; in one-stem notation it is written with the unusual "split stem". Finally, consider *f*, where the *noteheads* are now far enough apart to be vertically aligned, but one of the associated accidentals still has to be moved.

Figure 37. Two voices sharing a staff, two stems

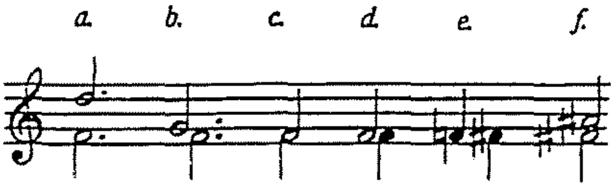


Figure 38. Two voices sharing a staff, one stem

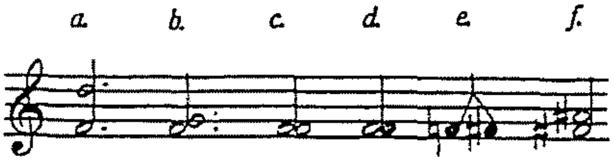
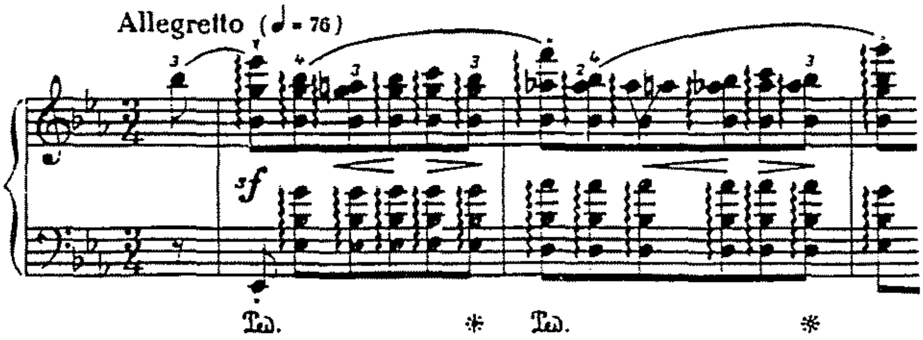


Figure 39. Chopin: *Etude*, Op. 10 No. 11.



All these problems and more occur in real music. Fig. 39 is an early example of a split stem, from the Chopin *Etude*, Op. 10 No. 11 (written about 1832). A more complicated situation is Fig. 40, from Bach's famous *Chaconne* (from *Partita No. 2 in D minor*) for solo violin. Here we have (at least in theory) four independent voices on one staff. In this case the only way to show the independence of the voices that have stems going up is to displace one note horizontally a bit. Since the same problem situation as in Fig. 37*b* is also involved, we end up with *three* different horizontal positions for a single point in time. Notice that the editor of this edition chose to position the augmentation dot on the top note between the stems of the next two lower notes, presumably to minimize the horizontal distances between the three positions. One might, of course, argue that (notwithstanding the way Bach wrote the manuscript) the voices are really not independent, since all but the top one have identical rhythms in these measures, and that therefore several noteheads could be put on one stem; this would alleviate the difficulties considerably. However, in the Brahms *Intermezzo*, Op. 117 No. 1 (Fig. 41), no such solution is possible. Here, the notation is only trying to show two independent voices per staff (each "voice" having a chord at this point), and these voices have totally different rhythms. The difficulties arise for three reasons. First, each chord involves notes on adjacent line and space positions, so two horizontal positions are involved within each chord. Second, the "normal", stem-direction-determining noteheads are on the left in the top staff and on the right in the bottom staff, and "the noteheads that determine stem direction [must be] vertically aligned" [ROSS70, p. 150]; thus the abnormal noteheads are on opposite sides of the aligned ones, and we are up to three horizontal positions. Finally, for reasons that are hard to put into words but obvious from the graphics, the eighth-notes synchronized with the chords cannot be put in any of these three positions, and we end up with *four* different horizontal positions for one temporal point.[33]

These last few examples are all cases where normal symbol positioning rules had to be modified to avoid "collisions". Avoiding collisions automatically is a difficult problem of computer graphics in general, and we will return to it in Sec. 5.2.1.

Figure 40. Bach: *Partita No. 2 in D minor for solo violin, Chaconne*, beginning.

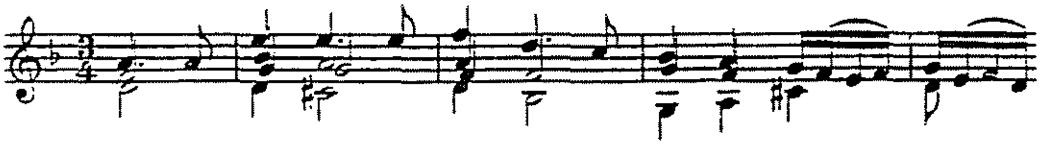


Figure 41. Brahms: *Intermezzo*, Op. 117 No. 1.

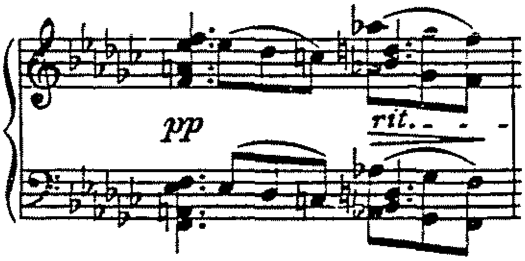


Figure 42. Berg: *Violin Concerto*, p. 1.

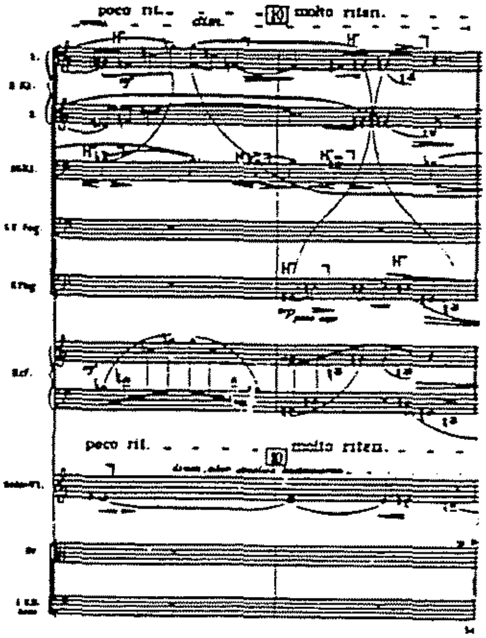
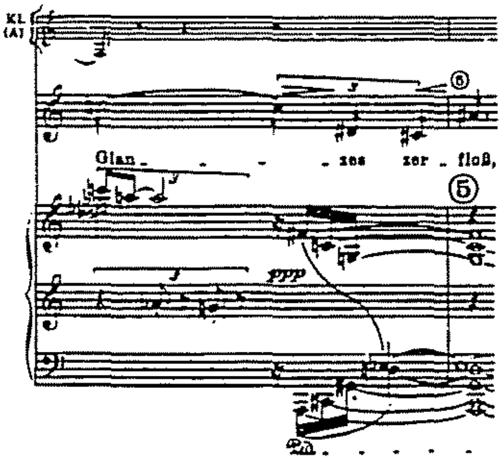


Figure 43. Schoenberg: *Pierrot Lunaire*, No. 9.



2.3.6.2.3. Multiple Staves for One Instrument. I have already mentioned a few simple instances of notation specific to a musical instrument or family of instruments. CMN is in fact loaded with instrument-specific features, some of which are highly complex. One of the most important examples is keyboard — piano, organ, harpsichord, etc. — notation, which, according to Read [READ69, p. 300] is "basically geared to the division of labor between the hands, with every notational element affected in some way by this consideration." [34] The most obvious feature of keyboard notation supporting the "division of labor" is, of course, the use of multiple staves — normally two, but occasionally three or even four — for a single instrument. One of the complications that can result from this automatic interdependence of multiple staves is a voice or voices crossing from one staff to another; another complication is stems with noteheads on more than one staff. Several of the "Counterexamples" below, e.g., Figs. 53, 54, and 59, illustrate these and related situations.

2.3.6.2.4. Multiple Voices as a Hierarchy. Multiple-voice music can be viewed most generally as involving a hierarchy. This structure is important because, generally speaking, the more closely related two voices are, the more strongly their notation can interact. This is the main reason that the slurs jumping staves in Fig. 42 (from the Berg *Violin Concerto*) are so surprising. Fig. 43 (from Schönberg's *Pierrot Lunaire*), where all three staves belong to one performer, is much less interesting. The hierarchy is easy to overlook because one hardly ever sees more than a few of its levels used simultaneously, but it is partly illustrated by a complex orchestral score like Fig. 44 (Beethoven, *Symphony No. 9*, I, p. 1). The possible levels, from outermost (most inclusive) to innermost (least inclusive), are[35]:

(1) Ensembles "to perform in different locations of the hall" [STON80, p. 172]. Relatively few compositions (the polychoral music of the 16th century Venetian School, Bartók's *Music for Strings, Percussion, and Celesta*, etc.) exploit this device.

(2) Families of instruments (brass, strings, etc.). Usually indicated by large square brackets in the left margin and/or interruptions in barlines.

Figure 44. Beethoven: *Symphony No. 9*, I, p. 1.

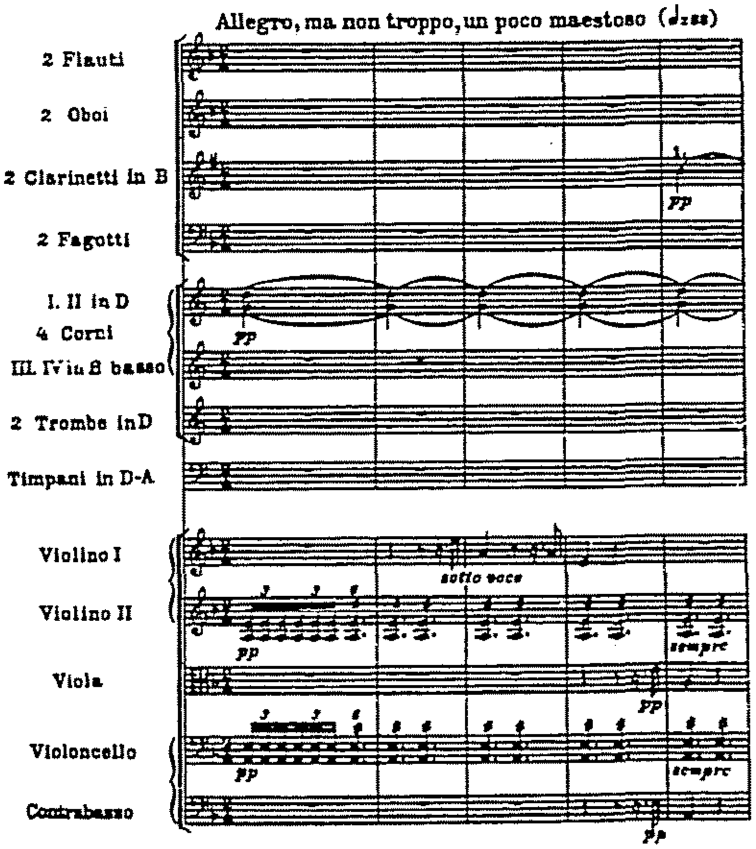
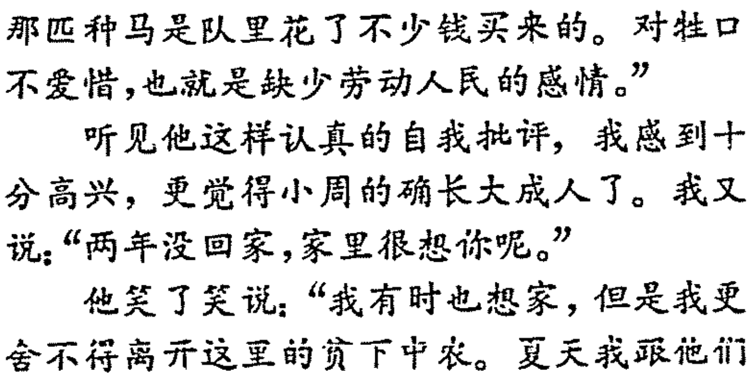


Figure 45. Chinese writing



那匹种马是队里花了不少钱买来的。对牲口不爱惜，也就是缺少劳动人民的感情。”

听见他这样认真的自我批评，我感到十分高兴，更觉得小周的确长大成人了。我又说：“两年没回家，家里很想你呢。”

他笑了笑说：“我有时也想家，但是我更舍不得离开这里的贫下中农。夏天我跟他们

(3) Sets of like instruments (horns, violins, etc.). Often indicated by small square or curly brackets in the left margin.

(4) Individual instruments. Most commonly, each occupies a single staff. Instruments that take more (e.g., piano, harp) have curly brackets in the left margin. When two or (rarely) more instruments share a staff, the rules for multiple voices on a staff given in Sec. 2.3.6.2.1 take effect.

(5) Voices within an instrument, whether on the same staff or not.

2.3.6.3. A Complication Slightly Related to Multiple Voices: Cues

An interesting notational problem arises when, in music for several performers, which obviously implies multiple voices, one performer has a long period of silence. Even though CMN conveys rest durations unambiguously, it is clear that the performer might make a mistake and resume playing at the wrong time. The notational device called the *cue*, an excerpt from another performer's part printed with smaller than usual symbols, is intended to alleviate the problem. [DONA63] comments:

> Any cue selected to aid in entry should be one that is easily heard by the player at rest. In many cases it may be the most prominent melodic line, but it need not necessarily be that; it might well be activity of immediately surrounding players. For example, if Trombone III has been resting for thirty-five measures and is to be cued for entry, the cue might be a slightly earlier entry of Trombones I and II if they, too, had been silent for a period.

2.3.6.4. Complications Unrelated to Voices: Page Layout

Two further complications in CMN, which I alluded to in the introduction to this chapter, have nothing to do with voices, but rather are matters of page layout. One results from a unique fact about music: it is an art that is usually performed with instruments that occupy the performers' hands, and often from written notation rather than from memory. The resulting complication is that of page turns. In the words of [READ69] (p. 442):

> Page turns in instrumental parts can make or break a work. Having to stop playing to turn a page can quite literally break the continuity of the music; the fact that the copyist has not provided for logical and accessible page

> turns can easily turn the performer against the music itself.

A comment by William A. Watkins, a commercial music engraver (in the loose sense of someone who sets music by computer), is even more to the point [WATK82]:

> . . . the enormous problem of [placing] page turns, particularly when there aren't any convenient measures [of] rest at which to place them . . . can involve knowing which notes can be played with only one hand by the particular instrument; ensemble seating arrangements (we once had to put an instruction in a clarinetist's part to turn the bassoonist's page); whether a passage is solo or tutti (in the event that the passage *must* be interrupted by a page turn); etc. I wouldn't care to try to write a program to encompass all of these things.

(Some parts, e.g., strings in an orchestra, are played largely by several performers in unison — "tutti" — but with occasional passages played by a single performer — "solo".)

As Lawrence Bodony has pointed out [BODO83], putting page turns in a conductor's score presents a quite different but probably equally difficult set of problems. The conductor always has an empty hand, so rests are not important, but she or he would rather not be confronted with numerous actions (e.g., cueing performers) to be taken instantly after a page turn.

Finally, there is a very strong tradition in CMN that the music should not end just anywhere in the middle of the last page, but should completely fill it. There is no obvious reason for this.

2.4. A COMPARISON OF NOTATION SYSTEMS

Is CMN really that complex? Is it more complex than, say, Chinese writing, or mathematical notation (to mention two systems with reputations for great complexity)?[38] This question is too vague to answer meaningfully unless we first pin down the meanings of the terms "Chinese writing" and "mathematical notation" as we have already pinned down "music notation" (by using the better-defined term "CMN", i.e., "conventional music notation"; see Sec. 2.2.2).

By "Chinese writing", or more generally any natural language writing, we will mean writing of ordinary text; we exclude tables, crossword puzzles, concrete poetry, and so on, which can extend the typographic characteristics of the language almost indefinitely. Similarly, "mathematical notation" herein will exclude geometric diagrams, graphs, and pictures like the ones commonly used in elementary calculus texts in discussions of volumes of rotation. These constraints are analogous to the constraint of restricting music notation to CMN, especially since purely pictorial representations have been used for music; see [STON80] for examples.

To answer our question, we also need to distinguish among *semantic* complexity, *syntactic* complexity, and *graphic* complexity. These three aspects — semantics, syntax, and graphics — correspond to points along a line going from most abstract to most concrete. The distinction between semantics and syntax is, of course, standard. Graphics or, let us say, typography, is concerned with mapping abstract objects (in English text, for example, the word "typography"), into concrete ones (for example, the two-dimensional visual pattern resulting from printing the characters "typog-" at the end of one line and the characters "raphy" at the beginning of the next, both in 10-point Palatino).[37] Typography is rarely considered along with semantics and syntax; however, much linguistics work is concerned with the auditory equivalent of this abstract-to-concrete mapping, namely phonetics. In the words of Hofstadter [HOFS83a],

> Syntax is a set of rules for mapping structures in a semantic space of high dimensionality down into a low-dimensional syntactic space. A syntactic space — usually one-dimensional — is still an abstract space, involving a sequence of concepts, not tangible or visible objects. Notation is a further mapping from the abstract syntactic space to the concrete, usually two-dimensional, typographic space. Syntax is concerned with conversion from time-independent semantic concepts to a time-dependent sequence of structures which are still abstract, i.e., nonvisual. The syntactic stage is a way station between the completely abstract semantic and the completely concrete graphic levels.

To clarify this, the rules for determining what character in Chinese can legally occur in a given context are obviously extremely complex (if indeed such rules exist, which is far from clear): this is a matter of both semantics and syntax. However, the rules that determine where and how to draw a Chinese character, once it is decided to use it, are quite simple: this is a matter of typography. To put it a different way, typography is what formatters and typesetters (whether human or computer) worry about. Since a notation is simply a domain-specific set of rules for representing syntactically ordered information in visual form (see Sec. 2.2.1), it should be obvious that typographic complexity is the relevant one here. Therefore the relative complexity of meaning conveyed by a page of Chinese or of mathematical or music notation need not be argued.

With these points settled, I think our initial questions ("Is CMN really that complex? Is it more complex than Chinese or mathematics?") are easily answered. Chinese is a good example to discuss first because its typographic complexity appears great (at least to most persons whose native language is English), but is in fact no greater than that of English — which is to say, not great at all. Written Chinese (Fig. 45, written in the recently-developed "simplified characters") involves some thousands of rather complicated characters; however, these characters are simply lined up side by side in lines or columns (both horizontal, as in the figure, and vertical arrangements are actually used), and when one line or column is filled, another is begun. Thus, a piece of text in either Chinese or English is conceptually a one-dimensional string of characters, all roughly the same size and chosen entirely from a predefined set. The complexities that text-formatting programs deal with, in either language, arise *almost entirely* from the practical requirement of breaking the text into numerous short segments to make it fit onto pages rather than scrolls.[38] The same cannot be said of the complexity of music or mathematical notation.

Fig. 46 includes two examples of mathematical notation that appear in the documentation for a well-known mathematics formatter [KERN75, KERN78a] to show the capabilities of the formatter. These two examples plus the set of simultaneous equations in Fig. 47 show a fair amount of complexity; let us compare the notation in them to the notation in Fig. 36 (the Ives page discussed previously). Neither is based entirely on a predefined character set: mathematics has variable-size braces of several kinds, fraction lines, radical signs, and such symbols as "$\int \sum \prod \bigcup \bigcap$"; music has note stems, staff braces of two kinds, slurs, groupet brackets, "hairpins", and glissando marks. Each has vertical-axis components that are intrinsic (i.e., not related to the dimensions of the writing surface) in two different respects:

(1) Each is composed of multiple horizontal lines of symbols where the vertical alignment of symbols across all lines is significant. In the set of equations it indicates correspondence among terms; in the music it indicates notes sounding at the same time.

Figure 46. Mathematical notation

$$G(z) = e^{\ln G(z)} = \exp\left(\sum_{k\geq 1} \frac{S_k z^k}{k}\right) = \prod_{k\geq 1} e^{S_k z^k/k} \tag{1}$$

$$= \left(1 + S_1 z + \frac{S_1^2 z^2}{2!} + \cdots\right)\left(1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^4}{2^2 \cdot 2!} + \cdots\right)\cdots$$

$$= \sum_{m\geq 0} \left(\sum_{\substack{k_1, k_2, \ldots, k_m \geq 0 \\ k_1 + 2k_2 + \cdots + mk_m = m}} \frac{S_1^{k_1}}{1^{k_1} k_1!} \frac{S_2^{k_2}}{2^{k_2} k_2!} \cdots \frac{S_m^{k_m}}{m^{k_m} k_m!} \right) z^m$$

$$\int \frac{dx}{ae^{mx} - be^{-mx}} = \begin{cases} \dfrac{1}{2m\sqrt{ab}} \log \dfrac{\sqrt{a}\, e^{mx} - \sqrt{b}}{\sqrt{a}\, e^{mx} + \sqrt{b}} \\ \dfrac{1}{m\sqrt{ab}} \tanh^{-1}\left(\dfrac{\sqrt{a}}{\sqrt{b}} e^{mx}\right) \\ \dfrac{-1}{m\sqrt{ab}} \coth^{-1}\left(\dfrac{\sqrt{a}}{\sqrt{b}} e^{mx}\right) \end{cases} \tag{2}$$

Figure 47. Mathematical notation

$$\begin{array}{lllll} & a_{12}x^{(2)} + & a_{13}x^{(3)} + & a_{14}x^{(4)} & = \lambda x^{(1)} \\ a_{21}x^{(1)} & & + a_{23}x^{(3)} + & a_{24}x^{(4)} & = \lambda x^{(2)} \\ a_{31}x^{(1)} + & a_{32}x^{(2)} & & + a_{34}x^{(4)} & = \lambda x^{(3)} \\ a_{41}x^{(1)} + & a_{42}x^{(2)} + & a_{43}x^{(3)} & & = \lambda x^{(4)} \end{array} \tag{3}$$

(2) Each has, *within* a single horizontal line, symbols whose meaning depends on their vertical position relative to some reference point. In the mathematics these symbols are, of course, the subscripts and superscripts. In the music they are, primarily, the noteheads, although other symbols' vertical positions can be significant.

These differences between natural language writing on the one hand, and music and mathematics on the other, are very deep. In a word, CMN and mathematical notation are fundamentally graphical, while natural language writing is only incidentally graphical.[39]

Is there, then, any difference in complexity between music and mathematical notation? Yes: music notation is more complex, for at least four reasons, all of which have already been mentioned; see Secs. 2.3.3.1.1, 2.3.6.2.1 and 2.3.6.4. For one thing, achieving the vertical alignment of the items in the horizontal lines in the mathematics is trivial: every hunk in every line must be aligned with a corresponding hunk (or at least a vacant spot, as in Fig. 47) in each of the other lines. In music, the relative horizontal positions of the hunks can be fairly complex, as in the Ives example previously discussed.

Second and more importantly, below the hunk level, the horizontal lines in mathematics are totally independent; vertically aligning the hunks is all that is necessary.[40] In contrast, horizontal lines (i.e., voices) in a musical score repeatedly interact at all levels, sometimes to a spectacular extent. (I have already discussed one important cause of this interdependence, namely lines on one staff that are close in pitch.) A simple example appears in Fig. 48 (Beethoven, *Symphony no. 9*, Eulenberg edition, III, p. 136) in the second, third, and fourth measures of the clarinet staff, where there are two voices on one staff, requiring each to make adjustments.

The third factor in the greater complexity of CMN is a question of character sets. As I have said, neither mathematics nor CMN is based entirely on a predefined character set, but in several respects CMN is much further from having such a basis. One of the non-predefined symbols in CMN — the slur — is more complex than anything I know of in mathematical notation. Slurs not only do not have a fixed size or orientation, they do not have a fixed *anything*, other than what might vaguely be called "slurriness". An outstanding demonstration of this fact is found in Maurice Ravel's *Le Tombeau de Couperin* (Fig. 49). Among other things, slurriness involves

Figure 48. Beethoven: *Symphony No. 9*, III, p. 136.

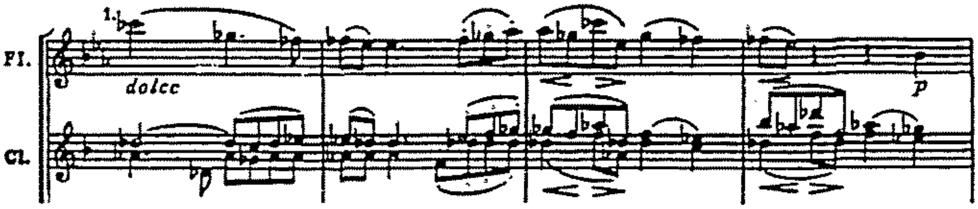


Figure 49. Ravel: *Le Tombeau de Couperin.*

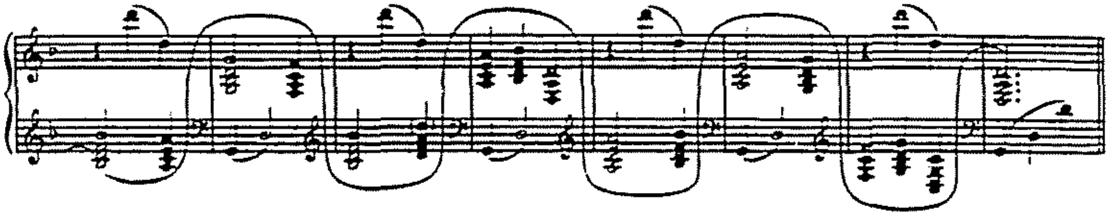
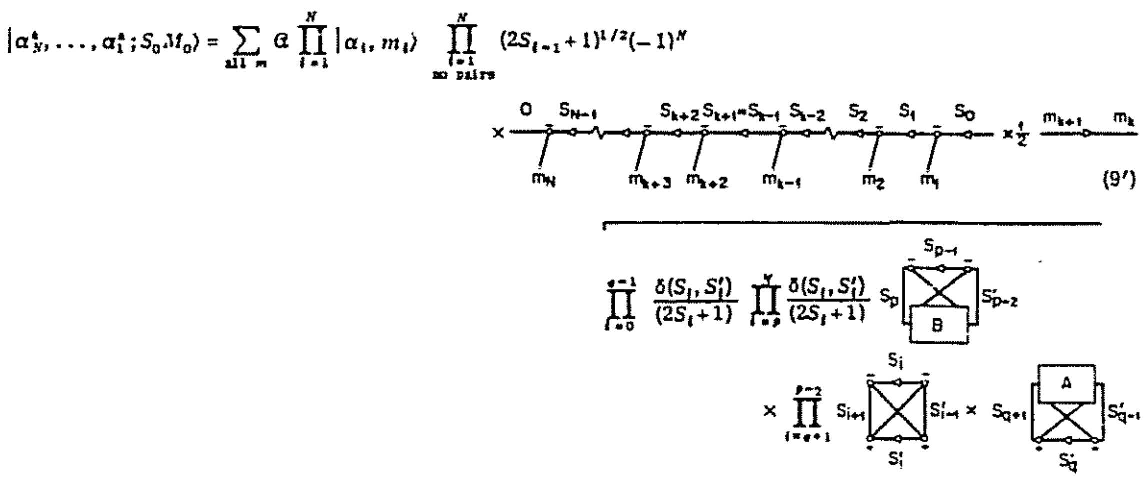


Figure 50. Mathematical physics notation



graceful curviness and forbids self-intersection. It not only does not involve a fixed shape, it does not involve a fixed or even a limited number of inflection points[41], as th: Ravel example shows. It also does not require continuity (see Figs. 42 and 55, the latter actually a tie), although "graceful curviness" probably implies several derivatives at every point of continuity.[42] This last comment touches on a deeper reason why CMN is further from a fixed character set than is mathematics: interruptibility is not a special feature of slurs in CMN, but is built into the system! In fact, almost any line-like symbol in CMN can interrupt another one — or they can intersect. Numerous examples of these phenomena are cited in Sec. 2.5. In view of all of this, it would not be an exaggeration to say that, in the ordinary sense of the term "character set", there is no such thing as a music character set. CMN does indeed have many characters — clefs, accidentals, and so on — in the narrowest sense of the term, but it also has symbols like the slur that can only be called characters in a very extended sense.

The fourth and final difference is related to the "practical requirement" of fitting the notation on pages instead of scrolls. In this respect mathematical notation is less demanding than natural language, since individual pieces of mathematics nearly always fit on a page. CMN, however, is more demanding than natural language. It has all the line-justification, running header and footer, etc., requirements, plus two more: page turns, and filling the last page.[43]

I might add one other observation. None of the terms "natural language writing", "mathematical notation", "CMN" is truly well-defined — all three have fuzzy boundaries, notwithstanding our efforts to pin their meanings down. Therefore the differences in complexity I have described are more a matter of practice than of theory. One could probably invent a piece of notation as complex as almost any CMN and argue with some force that it was legitimate mathematical notation[44]; but no existing mathematics formatting program — for example, Knuth's TEX [KNUT79], the UNIX "eqn" program [KERN78a] — deals with anything analogous to the above four problems, and I doubt if anyone notices the difference. All serious music formatters deal with at least the first of the four. None handles the other three fully, and many users are acutely aware of the lack.[45]

For more evidence of the trickiness of CMN, see the next section.

2.5. COUNTEREXAMPLES AND "FULLY AUTOMATIC HIGH QUALITY MUSIC NOTATION"

I have now presented evidence that CMN is more complex than such obvious rivals as Chinese writing or even mathematical notation. I have previously made strong claims (in Sec. 1.3) for the complexity of CMN in abstract terms, specifically that the correct formatting of a lot of music requires considerable intelligence (perhaps beyond the state of the art of artificial intelligence), and the correct formatting of a small amount of music probably requires human intelligence, specifically "common sense" — well-known bugaboo of artificial intelligence. I promised to give evidence in support of this assertion; it will be presented now, primarily through an annotated set of examples of published music. I call the items in this collection "counterexamples" because they are intended to counter the view that CMN, while it may have many complex details, is in principle easily mechanizable.

One plausible objection to this collection is that these examples really are not "conventional music notation", but involve nonstandard extensions to it. This is impossible to refute absolutely, since the boundaries of CMN — like everything else about it! — are rather ill-defined. In particular, music notation is an evolving system, so it might be claimed of any particular example that it includes notation that really precedes CMN or that is new and has not yet become CMN. In order to weaken such objections as much as possible, I have included only examples published by well-known music publishers and only from works of major composers, written within the 1700-to-1935 period I defined (in Sec. 2.2.2) as CMN. Composers represented include Bach, Bartók, Beethoven, Berg, Berlioz, Brahms, Chopin, Debussy, Franck, Grieg, Liszt, Mendelssohn, Mozart, Rachmaninoff, Ravel, Schönberg, Schubert, Schumann, Scriabin, R. Strauss, and Wagner. Note also that this collection is far from exhaustive: it is based on an examination of a minute fraction of the relevant musical literature.[46]

Another possible objection to many of these examples, particularly those classified as "Collisions", is that they are simply mistakes by the composer or editor and therefore show nothing at all about music notation. Again, this cannot be refuted totally, but in most cases the peculiar usage I am pointing out occurs several times in the movement. In any case, I think the examples support each other and tend as a group to undermine this objection: if similar "mistakes" are committed to paper over and over again by highly reputable composers and publishers, how can anyone insist that they are mistakes? Who makes the rules?

I want to emphasize that my point is not just that the rules of CMN are highly complex. Rather, I am making an argument about music formatting analogous to Y. Bar-Hillel's well-known argument [BARH60] about natural language translation: that what he calls "Fully Automatic High-Quality Translation" requires full artificial intelligence, i.e., human-level intelligence. What I call "Fully Automatic High-Quality Music Notation" (henceforth, "FAHQMN") requires human-level intelligence because:

(1) No set of rules that does not incorporate a high enough degree of flexibility to resolve a great variety of unanticipated conflicts can possibly handle all of CMN.

(2) The only way to get that much flexibility is with a system in which rules are not built-in but emerge from interactions on lower levels where competing forces can interact in parallel in such a way that superior solutions "float" to the top [HOFS83c].

(3) Finally, these lower-level pressures can involve the semantics of the music as well as common sense to an arbitrary extent.

The key word is "unanticipated": the variety of possible conflicts is so great that anticipating all situations is out of the question. Consider the Chopin *Nocturne* in Fig. 51. What imaginable algorithm would be able to position the beams in the upper staff as shown — a notation of doubtful "correctness", but probably the best that can be done with this music? (One could, of course, handle it with (in the words of [BARH60]) "a completely arbitrary and *ad hoc* procedure whose futility would show itself in the next example.")[47] A few more brief examples should drive the point home:

(1) Deciding where to place page turns can involve such nonobvious "world knowledge" as which notes can be played with one hand on the instrument in question and ensemble seating arrangements (Sec. 2.3.6.4). Deciding what notes to use as cues presents similar problems (Sec. 2.3.6.3).

(2) Deciding where to change clefs can involve a great deal of musical semantics. (These last two questions are strongly reminiscent of the text-formatting

Figure 51. Chopin: *Nocturne*, Op. 15 No. 2.

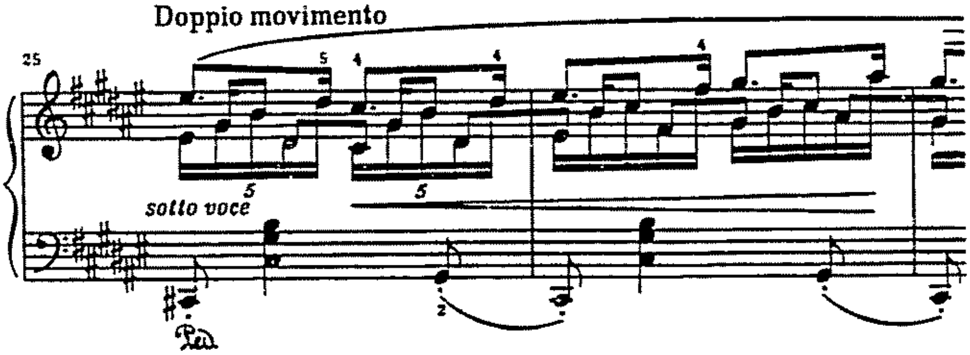


Figure 52. Scriabin: *Sonata No. 1*.

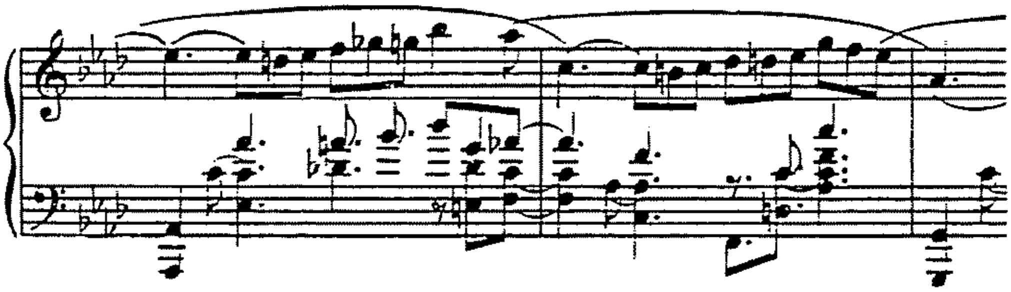
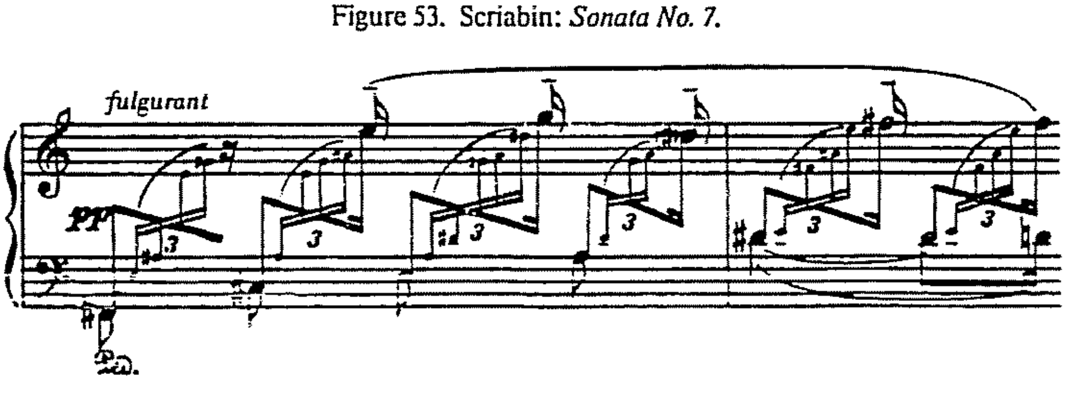


Figure 53. Scriabin: *Sonata No. 7*.



problem of avoiding "semantically bad" line breaks discussed by Knuth and Plass [KNUT81].)

(3) For the sake of clarity, composers and editors frequently write redundant accidentals, e.g., a flat in front of a pitch that is flatted in the key signature but that had a natural in the previous measure, as in Fig. 54. The rules for use of these "cautionary accidentals" given, e.g., by Stone [STON80, p. 55] are highly dependent on the semantics of music, especially on the complexity of the harmony. (All three of these points represent a different situation from the Chopin *Nocturne* in that the question here is one of choosing the best of several versions, all of which are absolutely standard notation.)

The question is one of what Hofstadter calls "slippability"; see Sec. 5.2.1.4.

I hasten to add that I do not claim that situations requiring commonsense judgements are as frequent in music formatting as in natural language translation; they probably are not. But they still occur reasonably often. So it is clear that writing the "ultimate" music notation system — one that does FAHQMN — would be a major problem in machine intelligence.[48] To my knowledge no one yet has even attempted to handle music notation in this way (nor does it make sense to do so in isolation from other problems).

Most of the examples in the following list are from piano music, which — because of its textural complexity — is an especially fruitful source of unusual notation. The examples are divided into several categories. Those in larger print with figure numbers are quoted here, the others are not. Bibliographic information is given in the format "(*year of composition*; *publisher/editor*, *year of publication*= *code*)". In many cases I have omitted one of the dates and/or the editor's name; nonetheless, it should be still be clear from the format what is what. *code* is used when a single publication is referred to more than twice. In this case full information is only given the first time, but a one- to three-character code is also given that is used in later references.[49]

A very interesting and somewhat similar set of examples, much less extensive but with detailed historical discussion, appears in the article "Notation" in *The New Grove Dictionary of Music and Musicians* [GROV80]. Another related set of examples appears in [MAXW83].[50]

(1) **Collisions**, i.e., cases where two symbols overlap or touch. See Sec. 5.2.1 for an extensive discussion of collisions and how a program might avoid them. An *intersection* is a collision of two symbols each of which is linear, i.e., conceptually one-dimensional. Intersections:

Chopin: *Nocturne*, Op. 15 No. 2 (Henle=H) (stems and beams in different voices): Fig. 51

Scriabin: *Sonata No. 1* (1892; International, 1976=I76), p. 9, 21 (slurs in same voice): Fig. 52

Scriabin: *Sonata No. 7* (1911; I76), p. 136 – 37 (grace note beams and regular note beams in the same voice): Fig. 53

See also:

Bach: *B-flat minor Prelude* from *The Well-Tempered Clavier*, Book I (1722; Kalmus/Bischoff=KB) (stem and tie, stem and beam)
Bartók: *Piano Sonata* (1926; Boosey and Hawkes), I, p. 9 (ties and stems)
Bartók: *Rhapsody for Piano and Orchestra*, Op. 1 (1904; Schirmer), p. 8 (hairpin across many stems)
Berg: *Piano Sonata*, Op. 1 (1908; Universal, 1926?=U26), p. 1, 4, 5 (slur and tie)
Brahms: *Rhapsody in G minor*, Op. 79 No. 2 (Peters/Sauer=PS) (hairpin and stems, hairpin and accidentals)
Brahms: *Capriccio*, Op. 76 No. 5 (PS), p. 18 (tie and stems in same voice)
Chopin: *Prelude in E minor*, Op. 28 No. 4 (1838; Schirmer/Joseffy=SJ) (slur and hairpin)
Debussy: *Reflets dans l'eau* (1905) (slurs in different voices)
Schumann: *Piano Concerto* (Eulenberg) I, 68, 72, 85-86 (slur and hairpin)
Scriabin: *Sonata No. 6* (I76), p. 111 (deliberate intersections of slurs and note modifiers)
Scriabin: *Sonata No. 6* (I76), p. 113 (slurs and stems in same voice)
Scriabin: *Sonata No. 9* (I76), p. 172 (groupet bracket and slur)

Intersections of barlines and hairpins, barlines and slurs, accidentals and slurs, and octave signs and slurs are too common to be worth listing. A few examples of collisions other than intersections (most of fairly common types):

Beethoven: *Piano Sonata*, Op. 106 (1819; Universal/Schenker, 1923), IV, p. 556 (notehead and trill)
Berg: *Piano Sonata* (U26), p. 11 (performance direction and hairpin)
Berlioz: *Symphonie Fantastique* (Eulenberg), V (stem and dynamic)
Brahms: *Capriccio*, Op. 76 No. 8 (PS), p. 29 (slur and character string)
Brahms: *Intermezzo*, Op. 117 No. 3 (PS), p. 81 (slur and clef)
Mendelssohn: *Violin Concerto*, I, m. 165 (fermata for one staff and another staff)
Ravel: *String Quartet* (1903; International), I, p. 19 (tie with key signature and time signature)

Schumann: *Piano Concerto* (Eulenberg), I, m. 248ff (stems and accents)
Scriabin: *Sonata No. 5* (I76), p. 80 (ledger line, expression mark, hairpin)
Strauss: *Also Sprach Zarathustra* (1896; Eulenberg=E), p. 25, 29 (dynamic and slur for different voices)
Wagner: *Tristan und Isolde, Liebestod* (1865; Kalmus), p. 29 (slur and performance direction)

(2) **Linear symbols interrupted by other symbols.** In every case I know of, the apparent reason is to avoid a collision. Interruptions of barlines are commonplace. A few examples are:

Beethoven: *Symphony No. 9* (1823; Eulenberg), I, p. 1 (barline interrupted by performance direction)
Mozart: *"Dissonant" Quartet*, K.465 (1785; Eulenberg), I, m. 21 (barline interrupted by dynamic)
Strauss: *Also Sprach Zarathustra* (E), p. 84 (barline interrupted by instrument number)
Wagner: *Tristan und Isolde, Prelude* (Kalmus), p. 5 (barlines interrupted by performance direction and dynamics)

Other symbols interrupted:

Berg: *Violin Concerto* (1935; Universal) (slur interrupted by notehead): Fig. 42

Debussy: *Danseuses de Delphes* from *Preludes*, Book I (1910; Durand=D) (primary beam interrupted by noteheads): Fig. 54

Franck: *Pièce Héroïque* (Durand), p. 26 (ties interrupted by notes): Fig. 55

Berg: *Piano Sonata* (U26), p. 7 (one tie intersecting stems and interrupted by rest)
Brahms: *Waltz*, Op. 39 No. 2 (International= I) (hairpin interrupted by slur)
Chopin: *Etude*, Op. 25 No. 6 (1836?; Paderewski/Dover) (hairpin interrupted by fingering)
Chopin: *Nocturne*, Op. 15 No. 3 (H) (ties interrupted by notes)
Ravel: *Le Tombeau de Couperin* (1917; Durand, 1918=D18), Prelude, p. 4 (primary beam interrupted by clef)
Scriabin: *Sonata No. 1* (I76), p. 9 (hairpin interrupted by beams)
Scriabin: *Sonata No. 6* (I76), p. 106 (ties interrupted by beams)

(3) **Symbols with nonstandard shapes or positions.** In most cases the reason is to avoid collisions; in some the symbol is being used for an unusual purpose. (It is interesting to observe that when a slur begins or ends with a tied note, the slur is — according to [READ69], [ROSS70], and [STON80] — supposed to include the entire tied series, but in practice very often does not. The apparent reason is to shorten the slur and thereby make collision avoidance easier. This

Figure 54. Debussy: *Danseuses de Delphes.*

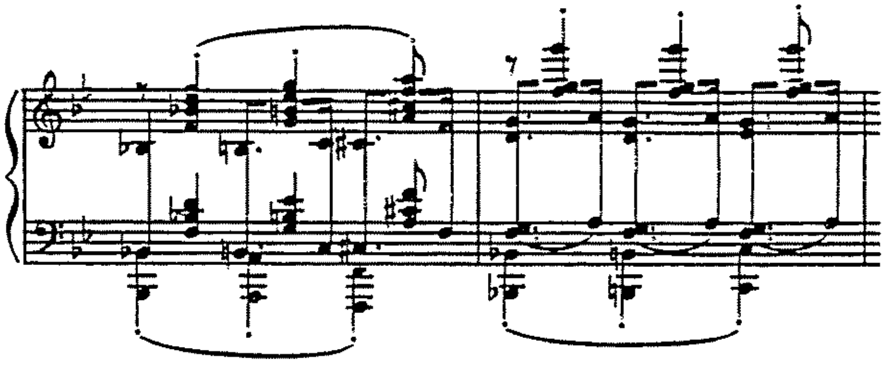


Figure 55. Franck: *Pièce Héroïque.*

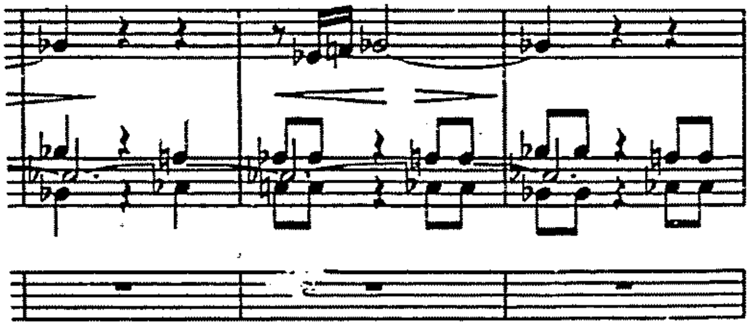
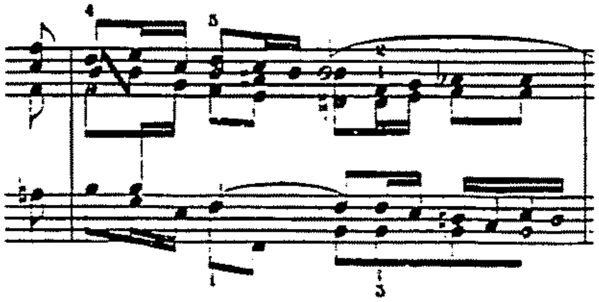


Figure 56. Bach: *B-flat Minor Prelude* from *The Well-Tempered Clavier*, Book I.



is one of the most glaring discrepancies between CMN theory and practice.)

Bach: *B-flat minor Prelude* from *The Well-Tempered Clavier*, Book I (KB) (two notes with the same vertical position connected with N-shaped beam): Fig. 56

Berg: *Violin Concerto* (1935; Universal) (slur jumping staves and performers; see [RAST82, p. 234]): Fig. 42

Chopin: *Etude*, Op. 10 No. 11 (1832?; Paderewski/Dover) (split stem, very unusual for the time): Fig. 39

Ravel: *Le Tombeau de Couperin* (D18), Minuet, p. 21 (slur with seven inflection points): Fig. 49

Schönberg: *Pierrot Lunaire* (1912; Universal) No. 9 ("Gebet an Pierrot"), m. 4, piano (slur jumping staves): Fig. 43

Bach-Hess: *Jesu, Joy of Man's Desiring* (Oxford, 1926) (noteheads on wrong side of stem)
Brahms: *Intermezzo*, Op. 119 No. 1 (PS) (stem only one space long)
Chopin: *Prelude in C Major*, Op. 28 No. 1 (SJ) (very short stems, thin beams)
Chopin: *Sonata No. 3* (Schirmer/Mikuli= SM), I, p. 66 (short stems, very thin beams)
Chopin: *Sonata No. 3* (SM), I, p. 70 (slur with two inflection points)
Rachmaninoff: *Prelude*, Op. 23 No. 4 (International) (slur with two inflection points)
Schubert: *Impromptu in E-flat*, Op. 90 No. 2 (Kalmus) (step-function octava sign)
Strauss: *Also Sprach Zarathustra* (E), p. 141 (bent trill)

(4) **Rhythm notation.** Caution is necessary here because until about 1750, rhythm notation conventions were considerably freer than later, especially for dotted notes. Violation of vertical (rhythmic) alignment (see Sec. 2.3.6.2.1 for discussion):

Bach: *Chaconne* from *Partita in D minor for Solo Violin* (c. 1720) (three stem and notehead positions for one attack time): Fig. 40

Brahms: *Intermezzo*, Op. 117 No. 1 (PS) (two stem, four notehead positions for one attack time): Fig. 41

Chopin: *Nocturne*, Op. 15 No. 2 (H) (one horizontal position — in fact one notehead — for regular and quintuplet 16th, for regular and triplet 16th): Figs. 51, 57

Bach: *C-sharp minor Fugue* from *The Well-Tempered Clavier*, Book I (KB) (three notehead positions for one attack time)

Figure 57. Chopin: *Nocturne*, Op. 15 No. 2.

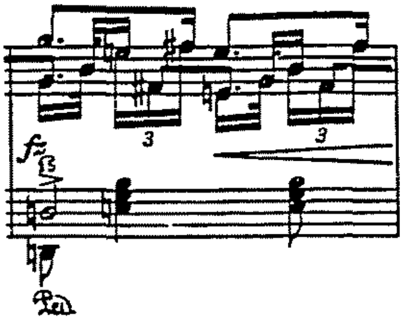


Figure 58. Bach: *Goldberg Variations*, No. 26.

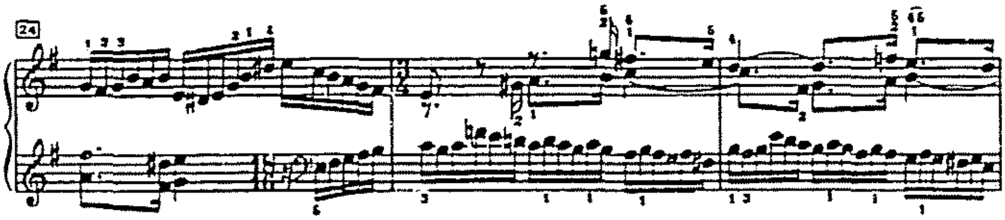
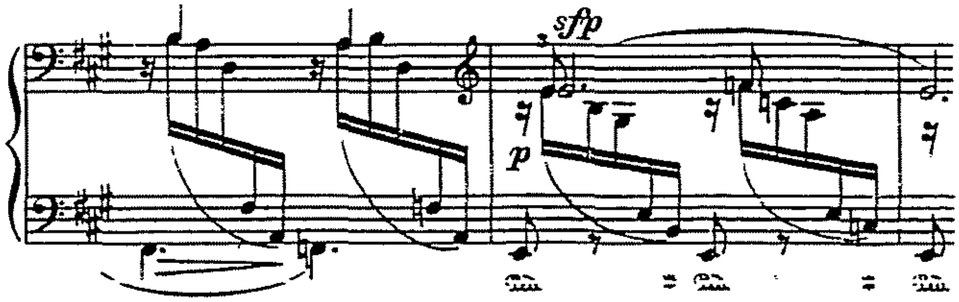


Figure 59. Brahms: *Capriccio*, Op. 76 No. 1.



Bach-Hess: *Jesu, Joy of Man's Desiring* (Oxford) (three stem positions for one attack time)
Debussy: *Les Collines d'Anacapri* (D) (three stem positions for one attack time)

Other rhythm notation:

Bach: *Goldberg Variations* (1742; Peters), No. 26 (change of meter in middle of measure): Fig. 58 (initially, the upper staff is in 18/16, the lower in 3/4)

Brahms: *Capriccio*, Op. 76 No. 1 (1879; PS) (written dotted half-note with an actual duration of eleven 16ths): Fig. 59 (in 6/8 time)

Grieg: *Albumblatt*, Op. 28 No. 2 (Schirmer) (groupet notation that is unambiguous but very unusual and evidently pointless: octuplet 16ths are identical to normal 32nds): Fig. 60

Beethoven: *Quartet No. 7 in F*, Op. 59 No. 1 (Breitkopf and Härtel), IV, pp. 29ff. (note before barline with dot after barline[51]): Fig. 61

Bach: *Cantata No. 21* (1713; Dover (Bach Gesellschaft)), p. 86 (note before barline with dot after barline)
Bach: *Mass in B Minor* (1732?; Barenreiter), *Symbolum Nicenum*, p. 140 (time signature of slashed "2", a holdover from 17th century practice)
Bach: *Prelude in D major* from *The Well-Tempered Clavier*, Book II (1744; KB) (double time signature of slashed "C", 12/8)
Beethoven: *Piano Sonata*, Op. 111 (Dover/Schenker), II, p. 604 (change of meter in middle of measure, hard to recognize because of unmarked triplets and because the new time signature should really be 3/8, not 6/16)
Brahms: *Trio in C minor*, Op. 101, III (double time signature of 3/4, 2/4)
Chopin: *Prelude in C Major*, Op. 28 No. 1 (SJ) (an instructive case: the right hand has the same rhythm almost throughout, initially notated precisely, but after a few measures oversimplified)
Chopin: *Prelude in D Major*, Op. 28 No. 5 (SJ) (eighth-notes starting a 16th-note before the barline)
Liszt: *Sonetto 104 del Petrarca* (unmarked groupets; "free" measures of much greater duration than time signature indicates)
Mozart: *Fantasy in C minor*, K. 475 (1785; Presser/Broder) (free measures, as above)
Rachmaninoff: *Prelude*, Op. 32 No. 3 (5 against 4 rhythm in grace notes)
Scriabin: *Sonata No. 10* (I76), pp. 198 – 99 (two time signature stacked vertically; dotted barlines sometimes followed by time signatures)
Schumann: *Symphony No. 1, "Spring"* (Kalmus), III, pp. 88, 102 (change of mter in middle of measure)

(5) **Miscellaneous.**

Debussy: *La danse de Puck* (D) (two clefs simultaneously active on one staff): Fig. 62

Figure 60. Grieg: *Albumblatt*, Op. 28 No. 2.

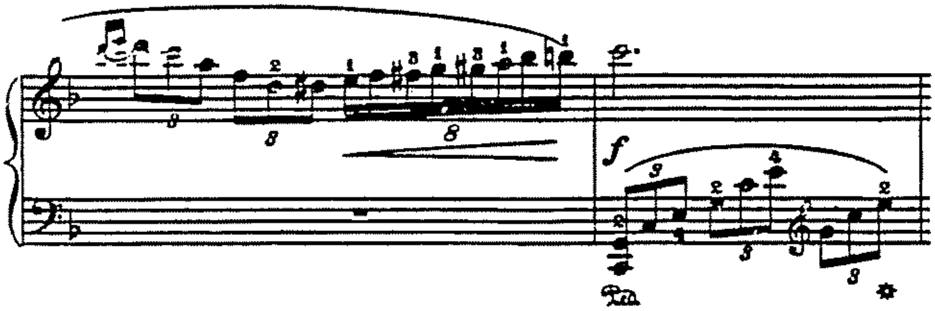


Figure 61. Beethoven: *Quartet No. 7 in F*, Op. 59 No. 1, IV.

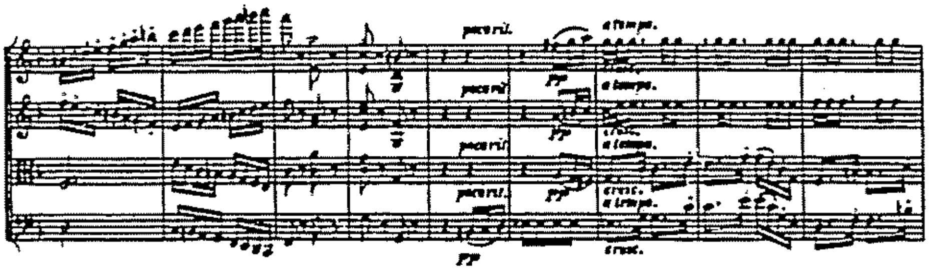
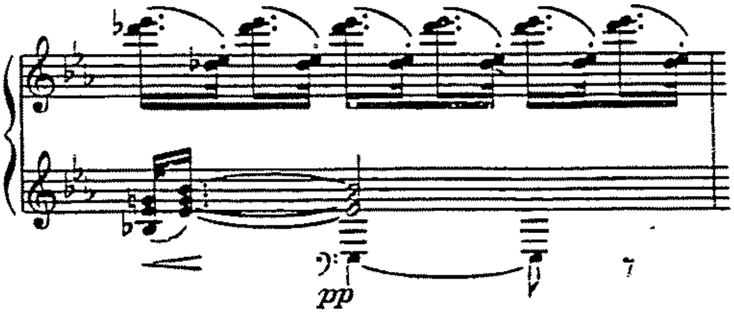


Figure 62. Debussy: *La Danse de Puck*.



Debussy: *Suite "Pour le Piano"* (1901; Durand), I (two clefs simultaneously active on one staff): Fig. 63

Debussy: *Les Collines d'Anacapri* (D) (8va and non-8va simultaneously on one staff): Fig. 64

Berg: *Piano Sonata* (U26), pp. 8 – 9 (hairpin across page break)
Brahms: *Ballade*, Op. 10 No. 3 (1856; I), p. 26 (two clefs simultaneously active on one staff)
Brahms: *Ballade*, Op. 10 No. 2 (I) (tie between notehead and beam)
Chopin: *Sonata No. 1* (SM), I, last measure (grace note with slur to fermata)
Debussy: *La Mer* (1905; Kalmus), II, p. 76, harp (beams in mid-air, accompanying a glissando mark to suggest its speed)
Debussy: *Suite Bergamasque* (1890; Fromont, 1910), I (two clefs simultaneously active on one staff)
Debussy: *Voiles* (D) (two clefs simultaneously active on one staff)
Ravel: *Le Tombeau de Couperin* (D18), *Forlane* (slur ending between staves in mid-air)
Scriabin: *Sonata No. 5* (I76), p. 90 (clef sharing horizontal space with note)
Scriabin: *Sonata No. 6* (I76), p. 106, 111 (tremolo involving note cluster with split stem, no part of which is vertical)
Strauss: *Also Sprach Zarathustra* (E), 1st cello, p. 13 (clef sharing horizontal space with note)

NOTES

[1] If the reader feels that timbre is somehow more complex than the other parameters, she or he is certainly correct. More elaborate classification schemes that subdivide timbre are frequently given, but for the study of CMN, they are quite unnecessary. An excellent recent study of timbre from an experimental psychologist's perspective is [GREY75].

[2] This situation is changing. Timbre, especially, plays a much larger role in music of the last 30 years than in earlier music. Music notation is responding by adding a large number of symbols to describe timbre. See [STON80] for the most authoritative discussion.

[3] A few of the better-known attempts are discussed briefly in [READ69] and in the article "Notation" in [GROV80].

[4] I refer here only to his instrumental works, not his "electro-acoustic" (electronic) ones.

[5] This is not to say that we should ignore the shortcomings of CMN for new music, simply that if an NMN ("new music notation") is needed, CMN is not shown thereby to be a bad system. Times change. For an excellent discussion, see [STON80].

Figure 63. Debussy: *Suite "Pour le Piano"*, I.

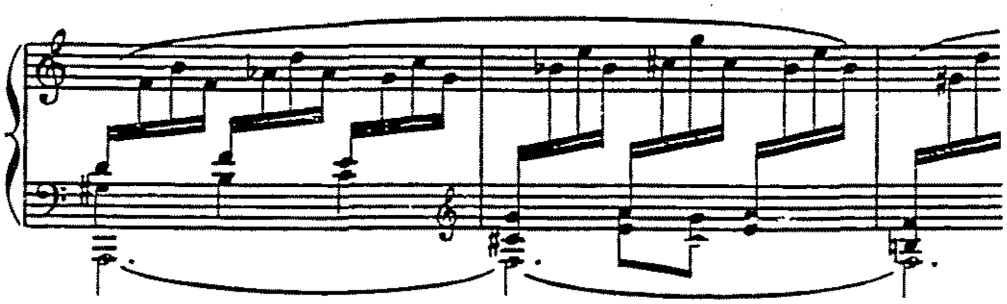
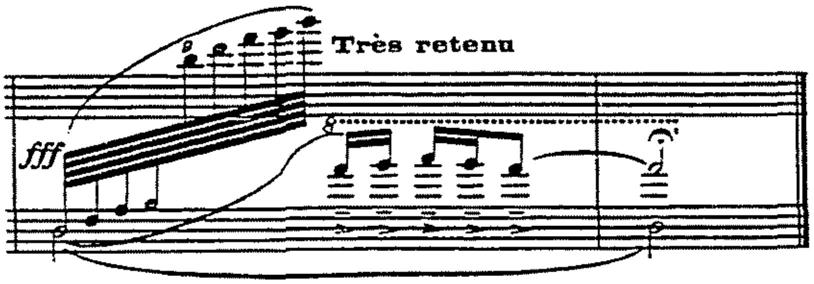


Figure 64. Debussy: *Les Collines d'Anacapri*.



[6] It is generally agreed that mensural notation, the previous notation system in Western art music, died out about 1600, but its last vestiges persisted well into the 17th century, so that 1700 is probably a safe starting point. See for example "Notation" in [APEL69]. On the other end of the period, after World War II, composers started inventing new notation at a rapid rate, so that 1935 is a relatively conservative ending point.

[7] Perhaps the most significant change took place around 1750: the semantics of the augmentation dot (Sec. 2.3.3.1) became clearly what they are now. Also, the modern scope rules for accidentals (Sec. 2.3.2) became fixed sometime between 1700 and 1750. A third change, in rules for clef usage, is mentioned in note 11 below. See [DART63] for a discussion of changes in the semantics of music notation, many very significant, since the Middle Ages.

[8] A standard dictionary of music, [APEL69], defines *notes* as "the signs with which music is written on a staff." This definition might charitably be described as half-hearted.

[9] Actually, parts frequently include "cues", brief excerpts from other performers' parts; see Sec. 2.3.6.3. Also, parts sometimes give the music for two or more related instruments, for example two flutes.

[10] This nomenclature agrees very well with perception. Two pitches any number of octaves apart do indeed sound very similar; the reasons for this are not fully understood.

[11] While these conventions have been rigid at any given point in recent musical history, they have not remained constant. For example, alto and soprano voice are both now written in treble clef. In the early 18th century, alto voice was written in alto clef, while soprano was written in soprano clef, which is now totally obsolete.

[12] This is an oversimplification. They are equally spaced in what is now the standard tuning system for nearly all Western music, "equal temperament". They are slightly unequally spaced in almost all of the many other tuning systems that have been used in various places and times. Even in Western music, the theoretically correct equal spacing is not always used: see note 13 below. For more details, see any text on musical acoustics, e.g., [BACK69].

[13] Again, this is not quite true. On an instrument that quantizes pitch as the piano does there can be no difference, but good string players, singers, etc., often perform these pairs of notes slightly differently. The difference between these *enharmonic* notes again has to do with tuning systems, and we will discuss it in Sec. 3.2.5. Again see [BACK69].

[14] This statement is not beyond criticism. For example, if a flat appears on an E4 in treble clef (bottom line of staff), is an E4 later in the same measure and staff in bass clef (one ledger line above the staff) also flatted? The standard texts say nothing about this. Also, many early 18th-century composers and many 20th-century composers have used the convention that accidentals are absolutely local and affect only the following note and immediate repetitions of it. However, we are concerned here with standard practice in CMN.

[15] A much rarer octave sign modifies pitch by two octaves; it differs from the usual one only by having "15ma" or "15" in place of the "8va" or "8".

[16] The notehead shapes shown are the usual ones, but other shapes with special meanings sometimes occur; see Sec. 2.3.5. Also, 64th-notes and rests are the shortest that are at all common, but 128ths and even 256ths have been used; see for example [READ69, p. 65, 117].

[17] [APEL69] says that the tie, "together with the bar line, is the most conspicuous achievement of modern notation over the earlier system of mensural notation, where it does not exist . . . Owing to the nonexistence of the tie in mensural notation, a note equalling five units was never used in duple meter . . . prior to *c.* 1600; only in triple meter could such a value be obtained, by subtracting one from six . . . "

[18] Actually, a triplet need not contain three notes or rests. It must simply contain notes and rests whose total duration is three times any basic duration. The triplet multiplies each item's duration by a factor of 2/3. For example, a triplet might contain two eighth-notes and two 16ths, for a total written duration of three eighths and total effective duration of two eighths.

[19] The term is not common; it appears in [DONA63]. Accessory numerals are often omitted when it is clear what is intended. For example, when 50 consecutive sets of triplet eighths occur, the "3" is invariably printed only for the first two or three. This is the situation with the quintuplets in Fig. 51. However, they are sometimes omitted in much more ambiguous contexts; see for example the second movement of the Beethoven Piano Sonata, Op. 111, and the Liszt *Sonetto 104 del Petrarca*, both referenced in Sec. 2.5, and the discussion of an example by Bartók in [STON80] (p. 82 – 84).

[20] The standard texts here are of varying usefulness. [ROSS70] is hopelessly confusing and ambiguous. [READ69] and [STON80] are better, though still not all one might want. Both recommend the "always-shrinking" approach. A book on 20th-century rhythm notation, [READ78], discusses this and related problems at great length.

[21] [READ69] discusses notes with one, two, and three dots only. However, there is at least one occurrence of a note with four dots in a well-known work, Hindemith's Symphony *Mathis der Maler* (introduction to the third movement).

[22] The one shown is given by $R(x) = 1/n$ if x is rational and $= m/n$ in lowest terms and n is a power of 2, else $R(x) = 0$.

[23] Most music theorists prefer the term *metric strength* for this concept. See also note 25 below.

[24] The only other possibilities that are clearly CMN are a large "C" (usually meaning 4/4), and a large "C" with a vertical line through it (usually meaning 2/2).

[25] Technically, the term for the rhythmic structure set up by the time signature and underlying the music is *metric structure*. This term helps theorists to distinguish between a composition's surface rhythm, which is explicit (i.e., audible) and varying, and its presumed background

rhythm, which is implicit and relatively constant.

[26] None of the standard texts quite makes this statement, but all of them imply it. See [READ69], p. 89; [ROSS70], p. 92 – 93; [STON80], p. 113. [DONA63] gives a more precise but not very accurate rule; see Sec. 4.4.2.

[27] Again, the standard texts are of uneven quality. All three make almost identically worded, vague statements: "the fractional beam must point toward the note of which it is a fraction" — but [STON80]'s examples are relatively good, while [READ69]'s are mediocre and [ROSS70]'s are totally unhelpful.

[28] These terms are generally used when the partial measure is no more than a single beat in length. It is not clear whether, strictly speaking, they apply to situations where the partial measure is longer than this.

[29] This dichotomy has been referred to as the difference between "descriptive" and "prescriptive" notation. CMN is almost always descriptive of pitch and time, is often descriptive and often prescriptive of loudness, and is usually prescriptive of timbre. See [SEEG58].

[30] The "+" marking, strictly speaking, means "left hand pizzicato", i.e., the string is plucked not with the hand that holds the bow, but with the other hand, the one that ordinarily presses down the strings to control pitch. The reasons for using the left hand are beyond the scope of this discussion.

[31] Actually, not all symbols fit neatly into one category. For example, tenuto marks may indicate a change in loudness as well as in time.

[32] There is no standard term for this concept. The term "hunk" was introduced by Gomberg [GOMB75, p. 60], although he does not define it clearly. [READ69] occasionally uses the term "structure" (e.g., p. 134). Knuth's idea in TEX of "boxes" [KNUT79] is quite similar, especially in that a box is formed by setting the "glue" connecting its subboxes, so that the contents of the box become rigidly emplaced. However, his boxes can be nested, while hunks cannot.

[33] In this case, the stem of the chord in the upper staff could have been pointed down, which would have reduced the number of horizontal positions to three, but one can invent a case where this could not easily be done — for example, just lower the pitch of the bottom note of the chord.

[34] The labor is divided among feet as well as hands in organ music: it normally has one staff for the feet in addition to two for the hands. (This is, of course, a footnote.)

[35] It is, unfortunately, nearly impossible to talk about this subject without using terminology that has strong connotations of either instrumental or vocal music. But I intend to imply no restriction: the same hierarchic structure occurs in all kinds of music.

[36] I do not claim that Chinese writing and mathematical notation are the most complex notation systems in existence besides CMN, but they are the most complex I am familiar enough with

to discuss. Other serious contenders might be Arabic writing, labanotation (a type of dance notation), and chemical notation.

[37] [FURU82] uses this abstract/concrete model extensively. See Sec. 5.3.

[38] Two complications dealt with by many high-grade text-formatting programs that have nothing to do with pages are *kerning* and *ligatures*; kerning is briefly discussed in note 40 below, and both are discussed more extensively in Sec. 5.2.1.1. While neither is trivial, they are both of relatively small complexity and importance. Hofstadter has pointed out an additional complication [HOFS83a]: Imagine a phrase in a language that is written from right to left, say Arabic, quoted inside a work in a language that is written from left to right such as English. If a line break is needed in the middle of the Arabic phrase, it is very unclear what should be done.

[39] Note, however, that the items I excluded from natural language writing at the beginning of this section — tables, crossword puzzles, and concrete poetry — all have the first of these "intrinsic vertical-axis components", and concrete poetry might reasonably have the second.

[40] Scott Kim has pointed out that this is not literally correct. If the expression P_n appears in isolation the *n* should probably be "kerned", i.e., moved under the bowl of the *P*, like this: P_n, while in a column vector that includes both P_n and Q_n as elements, the *n*'s should probably be aligned, so that the *n* could not be kerned. Again, this effect is trivial compared to what happens in music notation.

[41] The most common number of inflection points is, of course, zero. Inflection points nearly always occur when the slur moves from one staff to another.

[42] Some excellent discussion of the mathematics of pleasing curves is in [KNUT79], especially Part 1, pp. 22-27.

[43] Another way to look at this is in terms of dimensionality. A rigorous analysis of the dimensionality of any of these systems would probably be extremely difficult, but a few informal comments may be useful. Natural language writing involves one dimension for its succession of symbols, and something more, certainly less than a full dimension, for symbol choice. As we have seen, CMN is in concept four dimensional, the four dimensions being pitch, time, loudness, and timbre. In practice, only two — pitch and time — are represented to any great extent; on the other hand, one staff frequently includes more than one line, i.e., more than one set of several dimensions. Even with the vagueness of all of this, it appears that natural language writing involves fewer than and CMN considerably more than the two dimensions of the writing surface. The dimensionality of mathematical notation is still less clear.

[44] A case in point is Fig. 50, from the *Physical Review A* [DRAK77]. I showed it to one mathematician with the comment that this was very unusual mathematical notation, to which he immediately replied "Oh, but that's physics!" — the implication being that therefore it "didn't count".

[45] Why is music notation more complex than mathematical notation? This is a difficult question to answer, and not really germane to our purpose here; but one likely factor is their differing attitudes towards macros. Both notations have macros, but music notation allows only predefined ones (see Sec. 2.3.6.1), while mathematical notation allows user-defined macros, which provide a syntactically regular way to extend it.

[46] Contributions to this collection are solicited. I would like eventually to publish a much more comprehensive collection of this kind.

[47] In the words of Fanya Montalvo [MONT83], "generating CMN is an AI-complete problem". She coined the term "AI-complete" by analogy with "NP-complete" to suggest that there is a large class of problems that are not obviously related but whose solutions are in fact all equivalent to full understanding of human intelligence.

[48] See [HOFS79], especially its comments on the semantic depth of music (pp. 676 – 77), and [HOFS82b].

[49] It is, of course, not generally possible to tell from a single published edition whether the composer or the editor is responsible for a particular piece of notation. It would be interesting to investigate this question (presumably by the usual musicological techniques of looking at manuscript sources, comparing published editions where more than one exists, studying the composer's style, etc.).

[50] I am concerned here with CMN that is typographically unusual. It would also be interesting to collect CMN that is semantically unusual, for example by being *clearly* unplayable if taken literally (every classically-trained musician has seen plenty of music that, for one reason or another, *may* be unplayable). Such notation certainly exists, and in the works of major composers. For example, the *Organ Fantasy in G Major* by J.S. Bach includes a note that is below the range of all (known) organs. Another example: it is not too hard to find piano music, e.g., Schumann's *Dichterliebe* Postlude and Brahms' Op. 118 No. 1, Op. 76 Nos. 3 and 8, that includes swells on individual notes; of course there is no way to increase the loudness of a note on the piano once it is struck. For a brief discussion of the unplayable-notation phenomenon, see [RAST82], pp. 235ff.

[51] According to Read [READ69, p. 117], the practice of writing dotted notes with the notehead before the barline and the dot after it was widely prevalent in the 16th and 17th centuries. Read says "this practice has long since become obsolete." However, more recent users have included Bach and Beethoven (examples cited here), Brahms (cited by Read), and — a few years after our CMN period — Bartók (*Sonata for Unaccompanied Violin* (1944), fourth movement).

3
Background and Related Work

3.1. INTRODUCTION

The roots of automatic music setting, like those of most other recent technological achievements, go back much further than is usually thought. Buchner [BUCH78, pp. 23 – 24] mentions a "melograph" invented in the 1750's for recording performances of music, its primary application having been to the creation of mechanical musical instruments. He says, "Attempts to treat the melograph as a typewriter for music and nothing else did not succeed in practice."

Of course, it was not until the advent of the modern electronic digital computer that automatic music setting became practical. Scores, if not hundreds, of computer systems for music setting have been developed over the last 20-odd years, and I have been working on mine for more than 14. In this chapter I will summarize my work and the work of several other researchers on automating music setting from the 1950's to the present.[1]

One can view the task of music setting in the broadest possible sense as having three aspects. Arranged from most abstract to most concrete, they are:

(1) *Selecting:* deciding what symbols to print. One might assume that decisions as to what symbols to print are always determined by the semantics, and therefore are always explicitly made by the user. There are two reasons why this is not quite true. The principal one involves ***system breaks***, points where the music notation is broken into separate systems, determined by the positioning process described in the next paragraph but nearly always at barlines. The syntax of CMN is such that system breaks often cause new symbols, particularly slurs, to be generated. Less important, it is very convenient for the user to be able invoke high-level functions of a system that generate symbols to be printed, e.g., automatic decisions on what notes to beam together, or my system's automatic rhythmic clarification feature, which can replace one note with several (see Sec.

4.4.1). In any case, selecting the symbols to print is a relatively small part of the task.

(2) *Positioning:* deciding where to print the symbols and, for some symbols, deciding on their size, orientation, and/or shape. This aspect, frequently termed "formatting", is the central part of the task and involves formidable difficulties.

(3) *Printing* the symbols, i.e., being a "music typewriter" with no concern for the syntax or the semantics of CMN. (The music typewriter, incidentally, is a very real device: see Sec. 3.3.2.) This is relatively easy; the only real problems are related to device independence.

In describing what various systems do, I will sometimes refer back to these three aspects.

One aspect of the "musicians' problem" mentioned at the beginning of this dissertation is particularly relevant to CMN output systems, namely CMN input. (I mean here input of the information contained in CMN, not necessarily of the notation itself, although — with optical scanning — that is one possibility.) As I have already said, FAHQMN — Fully Automatic High-Quality Music Notation — is far beyond the state of the art of computer science today. One of the possible compromises (see Sec. 1.3.1) is to abandon full automation, i.e., to support some amount of interaction. In such a system one cannot always separate the input part of the system from the output. In any case, editing and formatting of any type of material are closely related activities, and input can even be considered a type of editing. Much confusion has been caused by authors (writing mostly about text processing) who have not been careful enough to make these distinctions. Since in this dissertation I concentrate on formatting, I will discuss music systems that are primarily input or editing systems only when they have formatting capabilities that are interesting in themselves. Otherwise, I may mention them, but will not go into detail.

The breakdown of work into input, editing, and formatting is natural to many problems in a variety of disciplines, although terminology varies so much from field to field that the similarities of the tasks are not usually noticed. I will discuss such questions in Sec. 5.3.

In this chapter, I assume the reader has a certain amount of background knowledge of computer graphics. For background information, see [FOLE82] or [NEWM79].

3.2. APPROACHES TO CMN INPUT

Before a program can print out music, it must obviously be given, or must compute for itself a precise description of that music in some form. What that form should be, however, is not so obvious. If the music is to be "fed in" by a person rather than composed by a program, the question is in some sense the inverse of the CMN output problem toward which the current work is primarily directed. I say "in some sense" because the *strict* inverse of CMN output would be direct pattern recognition of CMN by computer. Indeed, a fair amount of research has been done on just this problem. However, several alternatives exist: recognition of digitized sound; "menu selection"; alphanumeric character languages; and performances on *claviers* — piano-type keyboards — or other musical instrument input sections connected directly to a computer. The borders between these methods are not always clear-cut, especially between the last two. In my opinion, as of this writing, the only practical methods are menu selection and those somewhere along the spectrum between character representations and directly-connected instruments. I will now discuss all five "pure" methods as well as various compromises.

3.2.1. Pattern Recognition of CMN

Research on computer pattern recognition of CMN by optical means includes, to date, two dissertations: those of Dennis Pruslin [PRUS66] and David Prerau [summarized in PRER71]. Pruslin handled only an extremely limited subset of CMN in just one measure on two staves. Prerau handled a much more substantial subset of CMN, but still without tempo, dynamic, or phrase markings, with only one voice on a staff, and again on only two staves. It is interesting to note that little has been done in this area for more than ten years. One might infer that this is because the problem has been totally solved, but this is certainly not the case! The opposite view is much more accurate: the problem has been put aside by researchers as too difficult to be solved yet, especially in view of the fact that other methods are usable (and indeed preferable for some applications). This is so notwithstanding the optimism of a 1972 review of these two dissertations [KASS72]:

> . . . the logic of a machine that "reads" multiple parallel staffs bearing polylynear [the reviewer's written-musical analogue of "polyphonic"] printed music in at least one "fount" [sic] and size can be seen to be no further than another

> couple of M.I.T. dissertations away.

(Pruslin and Prerau were both at M.I.T.) This may be correct, but no further dissertations on the subject — from M.I.T. or elsewhere — have been forthcoming. The only later attack on the problem to my knowledge is that of Wittlich and Martin [WITT74], who worked on it until about 1976. In Wittlich's opinion, the project was not very successful. They attempted "right off the bat" to handle much more complex music than either Pruslin or Prerau, namely a full page of piano music by Debussy, complete with chords, dynamics, slurs, etc. Wittlich now feels that this was too ambitious, but that the main problem was the difficulty of obtaining high-quality microfilm for the optical scanner they used, which was of such high resolution it would "see" every blemish [WITT82].

In view of the tremendous strides technology has made in recent years, the time may be ripe for another attack on pattern recognition of CMN.

3.2.3. Recognition of Music from Continuous Digitized Sound

Another method that appears not to be practical yet is recognition of music and production of CMN for it from continuous sound. However, the situation here is very different, in that transcription of sound into CMN has been an active area of research in recent years. This task is frequently done manually by musicians under the name "taking dictation". For a computer, the task involves enormous signal-processing and psychoacoustics problems, somewhat similar to those of continuous speech recognition, especially with multi-voice music when more than one note is being played at a time (which, of course, is the usual case). Determining from continuous digitized sound even so basic a fact as a note's pitch can be amazingly difficult when one is dealing with the complex timbres of common musical instruments. Even when the pitches and durations of all the notes have been determined, the remaining work is far from trivial and, in fact, is identical in most respects to that of producing CMN from a performance on a directly-connected instrument; see Sec. 3.2.5. There is one significant difference between the two methods: low-level acoustic information from earlier phases that is not available with a directly-connected instrument may be useful in guiding high-level decisions made after this point of the process. We will see an example of this shortly.

The very first attack on this problem seems to have been that of James A. Moorer [MOOR75, MOOR77].[2] He described his work as follows:

> A piece of polyphonic [i.e., multivoice] musical sound is digitized and stored in the computer. A completely automatic procedure then takes the digitized waveform and produces a written manuscript which describes in classical musical notation what notes were played [via Leland Smith's MSS system]. We do not attempt to identify the instruments involved . . . It would appear that it is quite difficult to achieve human performance in taking musical dictation. To simplify the task, certain restrictions have been placed on the problem: (1) The pieces must have no more than two independent voices. (2) Vibrato and glissando must not be present. (3) Notes must be no shorter than 80 milliseconds. (4) The fundamental frequency of a note must not coincide with a harmonic of a simultaneously sounding note of a different frequency. The first three conditions are not inherent limitations in the procedures, but were done simply for convenience. The last condition would seem to require more study to determine the cues that human listeners use to distinguish, for example, notes at unison or octaves . . . In general, the system works tolerably well on the restricted class of musical sound.

Moorer also remarks that "[T]he process is extremely costly in terms of computer time." The basic method he used was the "heterodyne filter", actually a directed bank of contiguous sharp-cutoff bandpass filters, each of which is intended to pick up one partial. In a survey of techniques that he did not find useful, Moorer points out that, for musical sound, "we cannot rely on the presence of the fundamental, or on the hope that the fundamental will be stronger than the second harmonic." Thus, such obvious techniques as counting zero crossings are unworkable.

More recently, Piszczalski and Galler [PISZ77, PISZ81] "refined monophonic [i.e., single-voice] music transcription to a fine art" [MOOR82].

The most recent work I know of is an ambitious effort, still in progress as of this writing, by Chafe, Foster, and others [CHAF82, FOST82]. They are applying artificial-intelligence techniques, including some developed for analysis of speech and other non-musical audio information, to the problem as part of a project to develop (among other things) "an intelligent editor of digital audio". They take advantage of the availability of low-level (acoustic) information in a flexible way to guide high-level decisions: in the words of [FOST82], "The system makes decisions, but they are not necessarily final. The original sound is always retained and is reexamined in part from time to time to refine the hypothesis." Using such techniques they have implemented "higher-level processing methods [that] attempt to deduce the musical and notational contexts that have been supplied by hand in all of the previous systems." The musical and notational contexts referred to include the clef, key signature, enharmonic notation (see Sec. 3.2.5), and time signature.

3.2.3. Menu Selection

Input with the menu-selection method involves selecting CMN symbols from a menu and constructing a page of music by giving the computer positioning information for each symbol that requires it (some symbols in some contexts can go in only one place). The menu may be displayed on a CRT with selections made by any pointing device, as in several music editing systems — for example, Donald Cantor's early one and Rebecca Mercuri's more recent MANUSCRIPT system [MERC81] — or it may be in hardcopy form on the surface of a digitizing tablet, as in the editing system of Gary Wittlich and his associates [WITT78].[3] In both of these systems, feedback is given by displaying the music on the screen of a CRT while it is being constructed. Encoding a piece of music is done by moving the cursor to the menu to select a symbol, then (in most cases) moving the cursor to the desired portion of the music notation to deposit the symbol. This is repeated ad infinitum, although most systems allow depositing instances of the same character several times without returning to the menu in between. In a way, this is not much different from a character representation (see the next section) where the characters are musical symbols rather than those of, say, ASCII. The main difference is that with menu selection, position information is not included in the character code, but instead is given directly by pointing.

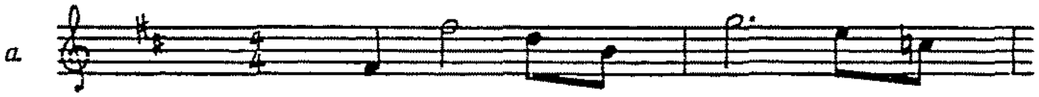
Jeffrey Hass has suggested a technique that combines the optical pattern recognition and menu selection approaches [HASS83]. He suggests that the user actually draw the music on a tablet and that a program recognize the symbols drawn and convert them to some code. Ideally, the recognizing program would provide feedback by displaying each symbol, appropriately cleaned up in both shape and position, as it is recognized. This is much easier to program than pure "pattern recognition" of the kind described in Sec. 3.2.1 because symbols are entered one at a time, so there is no problem of figuring out what is part of one symbol and what is not, and they are entered on line, so the program can ask for immediate clarification of an item if it needs it.[4]

3.2.4. Alphanumeric Representations

The simplest and most commonly used method is the one-dimensional alphanumeric or pure character representation. It requires no special hardware and is readily handled by conventional programming languages. Unfortunately, alphanumeric character notation is also the least satisfactory, because, as we saw in Chapter 2, CMN is very different from and much more complex than the natural language writing for which alphanumeric notations were originally developed. Numerous character-based "music input languages", all of which to my knowledge are regular languages (in the formal language theory sense of "regular"), have been developed; for descriptions of several typical ones see [BROO70]. Among the character-based languages discussed in that volume, two that deserve special mention here are DARMS (originally called the "Ford-Columbia language") [BAUE70, ERIC75, GOMB75] because it is the best-known, probably the most complete, and was originally developed for music printing; and MUSTRAN, because it is the one I use.

The DARMS music input language appeared in about 1965. It was the fruit of a project intended "to develop a code system that will enable us to print music under computer control by using currently available photo-composition equipment." [BAUE70] DARMS not only is exceptionally complete in covering music notation, but also appears to have good facilities for dealing with the extensive repetition often found in music notation. However, it has the (in my opinion) outrageously poor design feature of using letter names for durations and numbers for vertical positions — exactly contrary to the universal practice of musicians, since in practice vertical position means pitch. For example, in DARMS, "Q" means quarter-note, "E" eighth-note, "S" sixteenth-note, etc. For thirty-second-notes and all longer durations, the system is relatively mnemonic, but the "S" one would like for sixty-fourth is already taken, and in fact DARMS abandons any real attempt at being mnemonic for shorter durations as well. Fig. 1*a* is a simple example of CMN; the DARMS code for it is given in Fig. 1*b*. Sixty-fourth and shorter durations are relatively rare, so not having good mnemonics for them is not too serious, but reversing the obvious mnemonic representations for pitch and

Figure 1. DARMS and MUSTRAN



```
b.  !G   !K2#   !M4:4   2Q   9H    7E    5E   /   30H.   8E    6*E    /
c.  GS, K2*K, 4=4,    4F, 2F+, 8D+, 8B, /, 2G+., 8E+, 8NC+, /,
```

Figure 2. Rhythm fitting

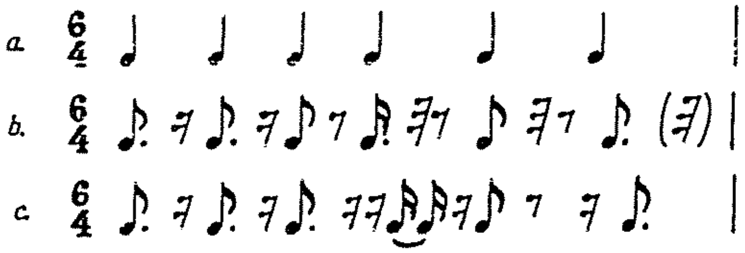
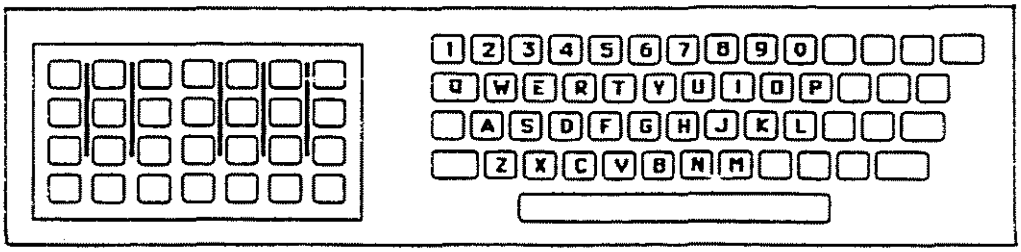


Figure 3. Dal Molin's keyboard



duration is surely one of the worst possible mistakes for a music input language. Imagine a programming language in which identifiers are numeric, while digits are represented by letters: instead of

```
AREA := 3.1416*R**2;
```

one would have to say something like

```
10 := T.OFOS*6**W;
```

— the absurdity of which is self-evident. The music notation equivalent is not *that* bad, but it is bad.[5]

MUSTRAN was developed by Jerome Wenker [WENK70, WENK74]. The MUSTRAN code for Fig. 1a is given in Fig. 1c. For a much more extensive example, see Sec. 4.8. Wenker originally developed MUSTRAN for ethnomusicological purposes, i.e., for encoding folk music of various origins, so that the original version lacked means of representing such art-music symbols as slurs, bowings, fingerings, etc. In MUSTRAN II, however, Wenker added a great deal of art-music notation; as a result, in terms of representing CMN completely, MUSTRAN II is probably second only to DARMS. In my opinion, MUSTRAN is also far more mnemonic than DARMS, and it has better software support — a translator and a library of utility routines — than any other alphanumeric language. On the other hand, the current translator is extremely batch-, and even punchcard-, oriented. Also, MUSTRAN's facilities for reducing repetitious coding are very limited. These facilities could, of course, be provided in a preprocessor, although almost certainly not as well as if they were integrated with the input language in a single processor.[6] This is especially true of use in an interactive mode, with feedback in CMN appearing on a screen as the music is being entered, which is highly desirable.

3.2.5. Directly-Connected Musical Instruments

By "directly-connected" musical instrument input, I mean that the channel transmitting information to the computer does not employ sound or its electronic equivalent (i.e., a signal which, amplified and fed into a loudspeaker, would ideally produce the same sound). Instead, the coupling is on a higher, more symbolic, level: the information the computer receives explicitly *says* what pitches are being sounded at what times by specifying one of a small repertoire of discrete possibilities. The difference is that between a player piano roll of a certain piece and a phonograph record of the same piece.[7] Loudness information is optional. Doing this is really practical only with instruments that inherently quantize pitch and time, particularly keyboard instruments. When it is done, however, the signal-processing and psychoacoustics problems mentioned in Sec. 3.2.2 are bypassed. (To date, the musical instrument used for direct-connection encoding has nearly always been a clavier: an electronic organ keyboard, sometimes sensitive to the velocity or pressure of the keystrokes in order to capture loudness information. The only other instrument whose use I know of is the guitar, input from which has recently been developed by New England Digital Corporation [NEDC83a].) Other nontrivial problems remain, for example, rhythm fitting, rhythm clarification, voice assignment, and enharmonic notation.

Jef Raskin has described the problem of *rhythm fitting,* also called *quantization*, as follows [RASK80]:

> Have a person using a metronome play on some instrument six quarter notes, in succession, at [the] tempo [♩ = 120, i.e., 50/100 second per quarter] ... The resulting data might well look like the data in [Table 1], which came from an experiment conducted with a push-button switch attached to my Apple II computer. The data was produced from the playing of an experienced musician and yet is irregular. There are two reasons the results from this very simple piece seem so ragged. First ... the actual duration of each note must be shorter than the indicated length in order to leave a short period of silence [between consecutive notes of the same pitch] ... Another reason lies in the normal variations in human motion ... The player was thinking of six equal notes, filling a measure as shown in Fig. 2*a*. But the computer ... received a series of rather irregular numbers. It would take some clever programming to determine that all of these notes were intended to be the same length. A moderately clever program might produce the music notation shown in Fig. 2*b*.

Actually, it is hard to imagine how a program would produce the transcription of Fig. 2*b*; what seems to me far more likely is the transcription of Fig. 2c. But both versions are equally undesirable, and for the same reason: they include details of the performance that not only were not intended by the performer, but seriously obscure what was intended — a series of notes of equal duration.[8]

An aspect of the rhythm fitting problem that has been almost totally ignored is that of automatically quantizing to groupets when appropriate. Nearly all

TABLE 1

Note	Starting time	Note length
1	0	32
2	53	34
3	101	37
4	167	22
5	210	28
6	268	30

All times are given in hundredths of a second.

systems simply generate a non-groupet approximation and require the user to edit the groupet notation in. As far as I know, the only method that has been described in print is my own, developed for the Wittlich et al music editor and described in [WITT78].

A problem related to rhythm fitting is *rhythm clarification*; it is discussed at length in Sec. 4.4.1 of this dissertation.

A very difficult but (in practice) less troublesome problem is that of correct *voice assignment*, i.e., deciding which notes belong to which voices.[9] One might be tempted to assume that this can be done simply by ordering the notes by pitch, that is, assigning the highest note to voice number 1, the next highest to voice number 2, etc. However, such an assumption is often wrong, especially in music for several performers: voices often cross.[10] Furthermore, the number of notes sounding at any one time may vary wildly over the length of a piece — in a single piano piece, often from zero to eight, and sometimes more. If fewer than the maximum number of notes are sounding at a given moment, how can one decide which voices are the active ones? Answering this undoubtedly requires far greater explicit knowledge of the semantics of music than anyone now possesses. Still another problem is that in some music, particularly piano music, a voice can include several simultaneous notes, i.e., chords (see Sec. 2.3.6). The developers of

Mockingbird studied the voice-assignment problem extensively and found it intractable, at least for piano music [MAXW83]. It is less troublesome than the rhythm problems discussed above only because it can be circumvented fairly easily, though at some cost to the user: by having her or him tell the system what notes belong to what voices. This can be done either as the music is being entered (for example, by encoding one voice per pass) or at a later time.

Still another problem is that of choosing the correct *enharmonic notation*, also called *note spelling*. Should a note be written as C-sharp or D-flat? To musicians, this is far from a trivial question; neither is it a trivial problem to decide from a keyboard performance which form to use. The depth of the problem is perhaps suggested by the fact that it has been studied extensively by a psychologist, H.C. Longuet-Higgins [LONG76]. He compares the relationship between C-sharp and D-flat to that between homonyms in natural language. Chafe et al [CHAF82] attack essentially the same problem with sound input in their system discussed in Sec. 3.2.2.

I have by no means covered all of the problems involved in transcribing music from what is basically a performance. As Raskin points out, "musical notation contains both more and less information than is contained in the performance"; many of the problems are consequences of this fact. See his article [RASK80] for a relatively nontechnical but thorough discussion. Another discussion, with some excellent examples of CMN for piano that would be very hard to produce automatically from clavier-type input, appears in the paper on the Mockingbird system mentioned before [MAXW83].

3.2.6. Real Music Systems and a Comparison of Input Methods

As I have pointed out, the distinction between two of the five approaches — the character and directly-connected instrument ones — is not at all clear-cut, and various compromises between them have had considerable success. Perhaps the most useful way to classify the compromise approaches is with two related dichotomies, one having to do with the encoding of time, the other with the encoding of pitch. There are encoding schemes where the entry of duration (and therefore rhythm) information is *proportional to real time*, and encoding schemes where it is not. I will henceforth abbreviate these descriptions to "real time" and "non-real time". Similarly, some encoding schemes in essence use position along a physical line — most often a line across the keyboard of a clavier — to represent pitch, and some do not. The former method might be termed by analogy "real pitch", the latter "non-real pitch". We may then say that an encoding method that is both real-time and real-pitch is definitely in the directly-connected instrument category; a method that is neither, definitely is not; and a method that is one but not the other, might or might not be. In actual usage, one combination — real time and non-real pitch — seems never to occur.[11]

An encoding system that uses real-time and real-pitch encoding has great advantages in that it reflects in a straightforward way both standard musical performance methods and CMN, but there are some tradeoffs to consider. Specifically, abandoning real time eliminates totally the rhythm-related problems discussed in Sec. 3.2.5, but introduces a major loss of efficiency for the many potential users who have spent thousands of hours learning to play the instrument in question fluently and, of course, in real time. We will return to the efficiency question shortly. Abandoning real pitch is harder to justify except on the purely pragmatic basis that alphanumeric keyboards with digital interfaces are much easier to find than claviers or other musical instruments with such interfaces. However, one can use an alphanumeric keyboard, simply interpreting the keys in an appropriate way: say, labelling consecutive keys in the middle row of the keyboard with white-note pitches and appropriate keys in the top row with black-note pitches. This is crude, but still fairly effective.

Table 2 gives the real-time and real-pitch status of a number of computer music input and setting systems. One other characteristic is indicated, namely whether the systems give immediate CMN feedback. "C" following the name of the system means that it can give CMN feedback, "S", that it can give feedback in spatial notation (CMN except for time notation; see Sec. 2.3.3.4). From a user interface standpoint, CMN input is like many other tasks in that well-designed feedback can make an otherwise awkward method of interacting with a system, such as alphanumeric music encoding, quite convenient. It should be noted that CMN feedback is distinct from CMN output not only conceptually, but usually also in practice. For feedback purposes, rapid response is much more important than high-quality printing; for final output, the reverse is true. I will say

TABLE 2

	Non-real time	Real time
Real pitch	Dal Molin (C) Scan-Note McLeyvier 1 NEDIT	McLeyvier 2 Mockingbird (S) Synclavier Wittlich
Non-real pitch	Kornfeld (C) MSS (C) MUSTRAN/SMUT NOTATE	

something about the hardware implications of this observation in Sec. 3.3.5.

Systems shown in the figure are Armando Dal Molin's system; David McLey's McLeyvier in two different modes; Dean Wallraff's NEDIT, developed at the MIT Experimental Music Studio [WALL78]; Kjær's Scan-Note system; William Kornfeld's LISP Machine music editor; my own MUSTRAN/SMUT system; Gary Nelson's NOTATE; Leland Smith's MSS; the Xerox PARC Mockingbird system; New England Digital Corporation's Synclavier system; Gary Wittlich and associates' system [WITT78]; and the University of Toronto Structured Sound Synthesis Project's Ludwig [REEV78]. All of the systems for which references were not given are discussed, with references, in Sec. 3.4.

Dal Molin's "Musicomp" terminal, like his older PCS-300, uses his "PCS" (Pitch-Character-Space) encoding technique. The terminal has a double keyboard, as shown in Fig. 3. For pitch information, it has an ingenious (and patented) variant of the clavier with four rows of seven keys each, corresponding to the white notes in four octaves of a clavier (an additional octave above and below is also encodable). This might be called "semi-real pitch". The division of the keyboard into four one-octave chunks is to facilitate touch typing. A conventional alphanumeric keyboard is used to specify the character (note, rest, or other). Each symbol is entered as a combination of a pitch, a character, and a following series of spaces (hence the "PCS" name), the number of spaces giving the item's duration — two for the shortest basic duration in the piece, three for the next shortest, etc. The device gives feedback in CMN on a CRT as the music is being entered.

All of the other real-pitch systems use conventional claviers. The McLeyvier offers the user a choice between real-time and non-real-time encoding; giving the user this option is very sensible, though the non-real-time technique it uses is (as of this writing) quite awkward.

Leland Smith's MSS system points up the importance of good feedback. With MSS, the user can sit at a graphics terminal typing her or his music in an alphanumeric language and see it immediately in CMN, so that errors — either semantic or typographic — can easily be spotted and corrections made on the spot. The method of making corrections also gives good feedback by indicating graphically what region is currently being manipulated. Smith comments [SMIT82]:

> I find that about 50% of my time is spent typing in the raw data and 50% is spent in screen editing. I have hopes of gradually changing this ratio to about 75%:25% but I doubt that it can get much better than this considering all the variable situations that can occur on a page of music.

One can get some feeling for the efficiency of various encoding techniques from the fact that it took an excellent pianist, Adrienne Gnidec, about five hours to encode the Chopin *Etude in C Major*, Op. 10 No. 1, in MUSTRAN. (See excerpt — about one-sixth of the piece — in Fig. 4, printed by my SMUT system from Gnidec's data. The left hand part is slightly simplified.) She could play the piece on the piano in less than two minutes, some 150 times faster. This comparison is, however, very misleading. First, the average encoded performance will include a few mistakes (by the performer or the rhythm-fitting routines or both), and in any case Raskin's dictum that a performance provides both more and less information than its notation applies; for both reasons, some additional time will usually have to be spent entering information in another way. An objection of another sort is that Gnidec could not have played the *Etude* in two minutes had she not already spent many hours practicing, not just the piano, but that specific

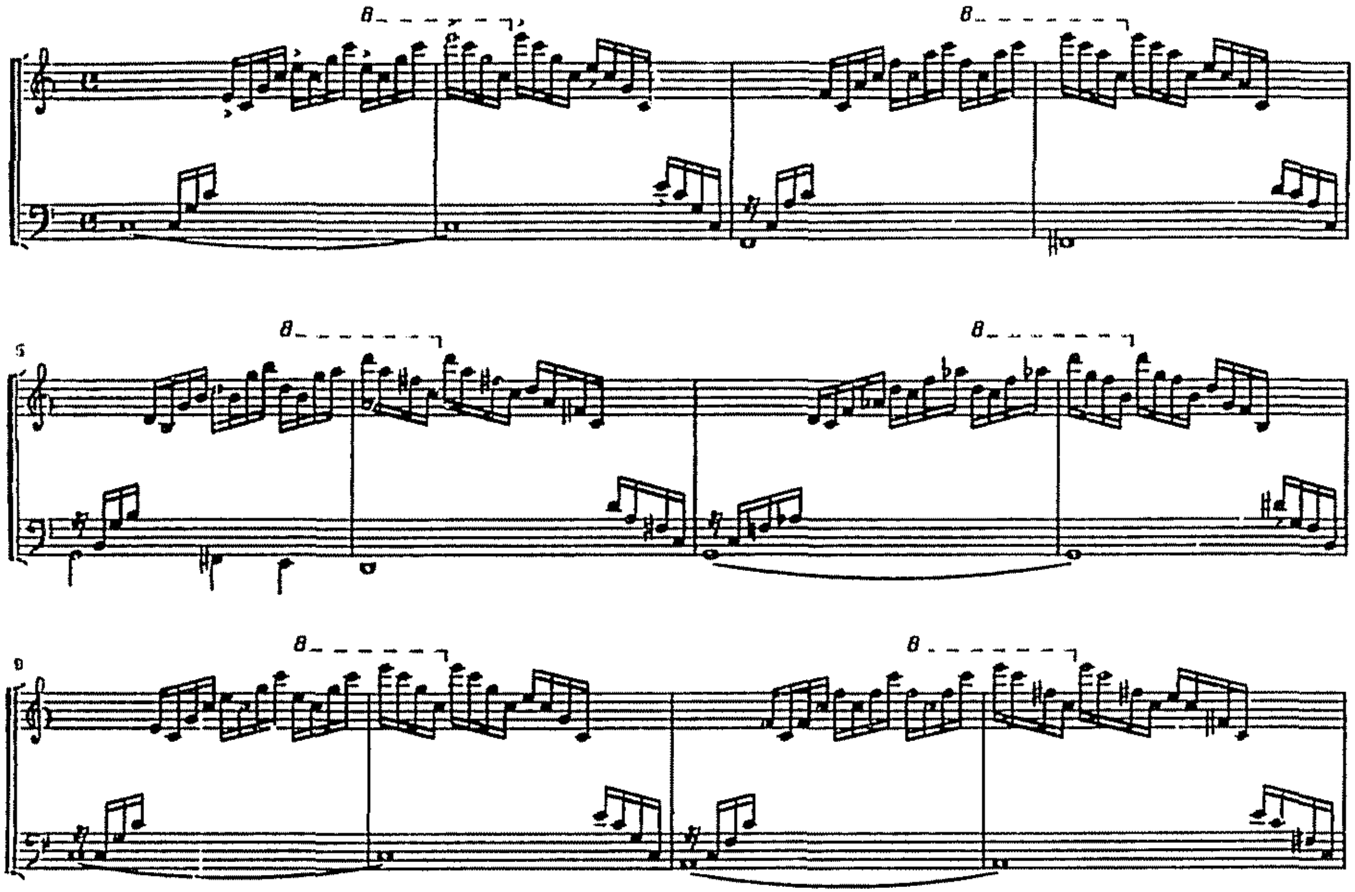


Figure 4. Chopin: *Etude in C Major*, Op. 10 No. 1 (set by SMUT).

piece. This is certainly true, although many musicians have already spent large amounts of time practicing pieces whose encoding is desired; this was in fact the case with Gnidec and the Chopin *Etude*.

The *Etude* is normally printed in four to six pages, so Gnidec's encoding took about an hour per page. Dal Molin estimated in 1975 [DALM75] that a page of music could be encoded in from 5 to 20 minutes in his "PCS" system on the terminal he was then using. This is probably close to the optimal speed for any encoding method.

3.3. APPROACHES TO CMN OUTPUT

The question of an approach to CMN output involves no difficult "human/machine interface" questions, as CMN input does. It is mostly a question of what hardware is used to produce the finished "original" pages of music, suitable for reproduction by conventional processes if desired. (Gomberg [GOMB75, pp. 26 – 29] discusses this question; however, I do not feel his comments are very accurate, particularly now that seven years have elapsed since he wrote them.) I will also comment here on hardware for CMN feedback, even though, strictly speaking, feedback is part of the input process; and I will make a few software-related comments.

The choice of output device is not as critical as it might on first glance appear. If a system is designed with reasonable concern for device independence, it can easily be modified to use another output device, at least of the same general type as it was designed for (and in a loose sense of "easily"). Thus the *category* of output device is much more important than the exact device. Also, systems that drive random-scan devices can nearly always be adapted to raster-scan without much trouble, but the converse is not true. See Secs. 3.3.5 and 3.3.6.

To give some perspective, I will begin by talking about three important noncomputer approaches to music setting. For more information on these methods and on the history of music setting, see Ross's book [ROSS70].

3.3.1. The Standard of Comparison: Plate Engraving

Traditionally, plate engraving has been the standard of comparison for music setting. Engraving is still generally considered the best music-setting method, but it requires a high degree of skill and is clearly a dying art. Most music published by major music publishers from the very early 1700's until very recently was engraved, although movable type was in widespread use for centuries and several other methods — pen-and-ink autography, transfer processes, music typewriter (Sec. 3.3.2) — have recently come into wide use. Strictly speaking, "engraving" is a misnomer, since music engravers do most of their work with sets of punches, reserving the "gravers" (the cutting tools, which have points of different sizes and shapes) for stems, hairpins, slurs, etc. A more accurate term for what music engravers do, occasionally used, is "plate punching". In any case, the engraver inscribes the musical symbols into the surface of a soft metal plate, the notation going from right to left (so that an impression will go from left to right). For examples of the music engravers' "character set" of punches, as well as a thorough discussion of plate engraving, see [ROSS70] (note, however, that Ross uses the term "engraving" in a highly generic sense for several different methods, and uses "plate engraving" to refer specifically to the use of gravers, punches, and metal plates).

3.3.2. Important Precursors: The Music Typewriter

Before discussing hardware for computer output of music, I would like to say a few things about an earlier and simpler approach to mechanizing the process of music setting, namely the *music typewriter*. Ross [ROSS70] discusses music typewriters at some length; the following is based on information he gives.

The first music typewriter was built around 1910. There are several kinds of music typewriters, but nearly all are actually modified conventional typewriters, to which they retain many similarities. Major differences are twofold: the mechanisms for positioning the carriage horizontally and vertically, and, of course, the character set. As we have seen, the positioning requirements music imposes on its symbols are far more complex along both axes than are the requirements of text. To accommodate this, different music typewriters have a variety of mechanisms, ranging from keys that move the carriage in tiny increments to (in one case) relying entirely on manual rotation of the platen. The character set of a typical music typewriter — the Effinger "Musicwriter" — is shown in Fig. 5; it contains 88 characters. It can use paper with preprinted staves, or staves can be typed with the character consisting of five horizontal lines. Notice that some large symbols such as the treble and bass clef are broken into pieces. Symbols such as slurs and nonhorizontal beams are not provided, even in sections; these must be added

Figure 5. Music typewriter character set

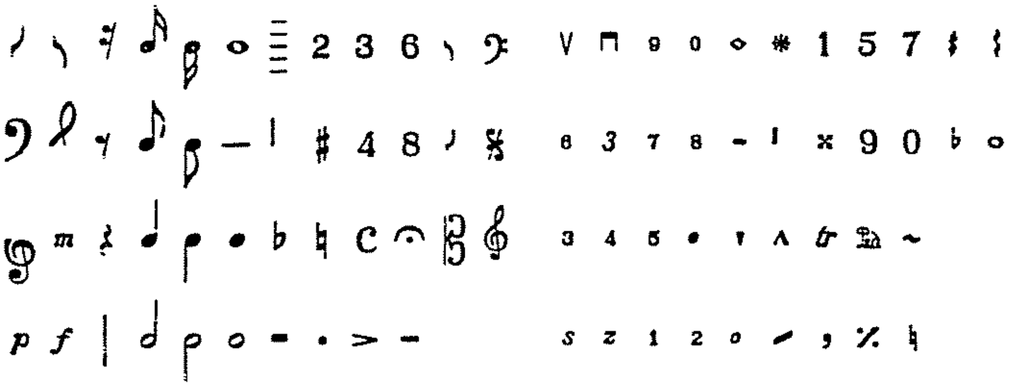
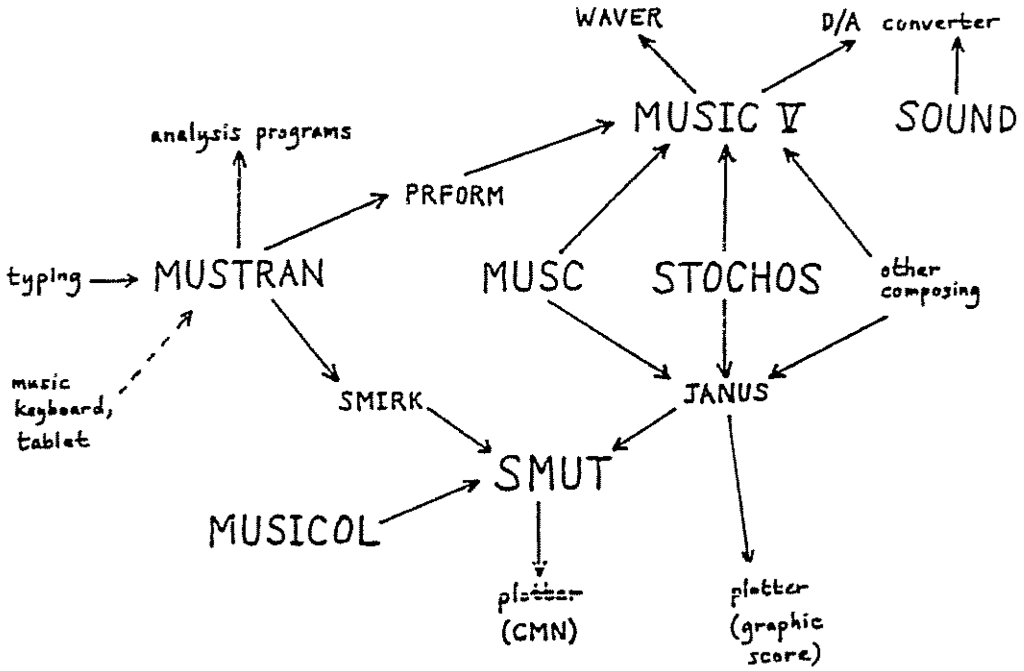


Figure 6. The Indiana University Computer Music System



manually. (Some music typewriters do support slurs in a crude way, by providing several different arcs that may be used to begin and end slurs. Apparently, one then types a horizontal line to connect them.)

At first glance, it may appear that music typewriters are very similar to ordinary typewriters in that both automate the same low-level portion of the setting task, namely (in terms of my summary at the beginning of this chapter) Aspect 3: the actual printing of the symbols. In actuality, conventional typewriters also handle automatically (with their automatic forward spacing) or semi-automatically (with the carriage return, tabs, margin stops, and margin bell) a significant fraction of the simple character positioning required in their domain, while music typewriters cannot handle automatically *any* of the very complex positioning their domain requires. Furthermore, conventional typewriters can actually print all of the characters needed for tasks of fair complexity, while music typewriters are lacking (especially in slurs and beams) for all jobs but the simplest. Thus, conventional typewriters automate all of Aspect 3 and much of Aspect 2, while music typewriters do not even automate all of Aspect 3. The seriousness of these limitations is attested to by the fact that most musicians, composers included, have never even seen a music typewriter.[12] In Ross' words,

> ... any professional engraver knows that a good hand-copier can put out a page of music at least twice as fast as any machinist ... It is elementary that much experience is required — beyond regular musicianship — to achieve proper notation. In short, one would be well advised to put no stock in salesmen's claims that their machines are speedy, require little practice and effort, and produce high quality work, all in one.

3.3.3. Important Precursors: Movable Type

In movable type printing, as with the music typewriter, each character appears on a separate piece of metal or other material. The form of the character may either be raised above, in the same plane as, or lowered below the surface; all three methods have been used. In any case things are arranged so that, when the piece is pressed against an inked surface, ink will be picked up in the right places. Printing from movable type was in heavy use for text from the time of Gutenberg, around 1450, until the rise in recent years of the phototypesetter (see Sec. 3.3.4). One might therefore guess that someone has tried printing music from movable type, especially since many of the difficulties of printing music with a fixed character set are not obvious on first consideration. This guess would be correct. In fact, music was first set with movable type in the late 15th century [ROSS70]. Soon thereafter it became the dominant method of music setting, a position it held until the rise of plate engraving (Sec. 3.3.1). A modern font may contain from 400 to 500 characters; it supports slurs and beams in the same "crude way" as do music typewriters, discussed in the previous section. Since a satisfactory alternative — plate engraving — became generally available, around 1700, the mechanical problems of getting numerous tiny pieces of metal to fit together well, in addition to the character-set-related problems I have already discussed, have discouraged the use of movable type except under unusually favorable circumstances. Such circumstances have generally involved relatively simple music with extensive text on the same page. For instance, hymnals and musical examples in encyclopedias have often been printed with movable type. For many years, all of the musical examples in the *Encyclopedia Brittanica* were set this way.

Movable type differs from music typewriters in allowing a much larger character set — a significant advantage — but disallowing overprinting — a major disadvantage. Limited though music typewriters are, movable type printing is for most purposes just as bad.[13]

3.3.4. Modern Character-set Devices

The hardware that has been used for computer printing of music might be classified broadly into general graphics devices with no built-in orientation towards music, and devices with "music character sets". I argued in an unpublished paper in 1977 [BYRD77b] that "there really is no such thing as a machine-independent music character set"; this attitude certainly fits my thesis, in Sec. 2.4 of the current dissertation, that CMN is deeply graphical in nature. Nonetheless, as the previous two sections have shown, extensive use has been made of music-setting methods involving devices with a limited number of fixed characters. In computer-based efforts, basically two such devices have been used: the *fully-formed character* (i.e., non-dot-matrix) *impact printer* equipped with a special *music character set*, and the *phototypesetter* (often called simply *typesetter*).

The impact printer has on the order of a hundred music characters, which it can overprint. This is rather like the music typewriter except for one major handicap: the impact printer can position characters only at grid points of a rather low-resolution grid, while the music typewriter has fairly high resolution both horizontally and vertically. Considering that music typewriters already have serious problems with such common symbols as slurs and nonhorizontal beams, it is clear that one should not expect much from the fully-formed character impact printer.

The other character set device — the phototypesetter — is vastly more flexible, and indeed has already produced very high-quality music notation. For the last 20 years the phototypesetter has been the standard machine for setting books, magazines, and newspapers. A detailed discussion of phototypesetters is far beyond our scope here. Suffice it to say that there are two kinds, called *photomechanical* (the traditional kind) and *digital* (which have been in widespread use for only a few years). Both kinds produce output on film, not plain paper, and both have their characters organized into fonts of a hundred or so each. (For music setting, of course, special fonts are needed.) Only photomechanical typesetters have been used for music setting thus far; I will discuss them first. Photomechanical typesetters work by shining light through templates of the desired characters, one after another; a complicated optical system, usually combined with motion of the film, then enlarges or reduces the image and deflects it to the proper place on the film. These devices have relatively large character sets — perhaps one or two thousand characters — which they can position on a page with fairly high resolution — typically .002 inch or so — and print with very high quality. In general, however, this is still not enough to draw really good slurs and beams, and in both of the projects described below that use phototypesetters (those of Dal Molin and Watkins) slurs and beams are added by hand, at least in some cases. Gomberg criticizes phototypesetters on one additional ground, namely that very large characters such as the staves must be assembled from smaller characters and that these machines do not have good enough repeatability to do so satisfactorily. I believe this judgment is now overly harsh, thanks to recent advances in technology, especially the advent of digital typesetters.

[SEYB83] surveys the state of the art in digital typesetters. Digital typesetters are superior to photomechanical ones in nearly all ways except that photomechanical ones are available for a lower cost. As a result digital machines have virtually taken over in all applications except those where the cost of the typesetter is crucial. Digital typesetters are extremely high-resolution programmable-character-set devices. Many models can have hundreds of fonts, hence tens of thousands of characters, on line at once: the only limitation is available mass storage. Some expose the film with lasers, while most use CRTs. Typical resolution is 1000 points per inch across a width of 8 inches or more.[14] (The CRT-based typesetters achieve far greater resolution than CRT-based display terminals are capable of by employing optical systems something like those of the photomechanical typesetters, so that the image on the CRT at any moment corresponds to only a small part of the output page.) Most digital phototypesetters make images with a method that is a hybrid between raster- and random-scan techniques: they scan the image area in a raster pattern, but rather than the (ideally zero-diameter) dots that are ordinarily used in raster graphics, they write variable-length parallel strokes, and they generally do not spend time traversing blank areas. Digital typesetters show promise of developing into extremely high-quality general graphics output devices ideal for music setting; the main thing lacking in current models is a general-purpose interface (current models have very specialized interfaces designed for typesetting text with limited graphics of the kind commonly encountered in newspaper and magazine text).

An obvious disadvantage of all devices with characters in hardware, including photomechanical typesetters, is lack of flexibility: the character set cannot be changed easily. An advantage of these devices is "offloading the CPU": all a computer needs to do to make such a device produce a character is to tell it which character and where, which can save a substantial amount of computation. Programmable-character-set devices like the digital phototypesetter offload the CPU without sacrificing any flexibility, and are therefore very attractive. A still newer type of hardware that has these advantages is the *laser printer*. Devices of this type have much in common with digital typesetters. The laser printer has major advantages in convenience, since it uses plain paper instead of film, and lower cost. It has two main disadvantages: lower resolution (current commercially available models have resolutions of at most 480 points per inch) and, in my experience, poorer and less consistent contrast.

3.3.5. General Graphics Devices

The subject of graphics hardware in general is a vast one. I will make only a very few general comments, plus some specific to music notation, here. For more background including definitions of the terms I use, see [FOLE82], especially Chapter 3, or [NEWM79].

For our purposes, two of the standard ways of classifying graphics output devices are useful: they may be *random-scan* (also called *vector-drawing*, *stroke*, or *calligraphic*) or *raster-scan*, and they may be *hardcopy* or *display* (non-hardcopy).

In general, random-scan devices, either display or hardcopy, have one major advantage: they can have much higher resolution than raster-scan devices.[15] They have some major disadvantages. They are severely limited in availability of colors or levels of gray scale. Also, random-scan displays have flicker problems when the image to be displayed gets too complex, and random-scan hardcopy devices are (in most applications) much slower than raster devices. Either type of display can produce a complicated image on a screen in a few seconds or less.

Now, how does all of this apply to music setting? I have already observed that a computer music-setting system may need CMN for two distinct purposes, feedback and final output. The number of colors or gray levels available is irrelevant for final output, since CMN uses neither color nor gray scale. For feedback, especially in editing, some way of distinguishing part of an image from another part can be extremely useful in any domain, including CMN; this can be done with color or gray level, or with other mechanisms — for example, reverse video or blinking.[16] For final output, hardcopy is required, and resolution is usually paramount, so the best random-scan devices are preferable to any raster-scan device. For feedback, a display device is obviously more convenient than hardcopy, and both types of display are quite fast, so the response-time limit of the hardware is not an issue; but flicker is. In terms of flicker, music notation is unusually demanding (an observation, incidentally, I have not seen made before).

On a random-scan display, flicker is, of course, always a potential problem. It is not ordinarily mentioned as a serious problem of raster displays, and, generally speaking, it is not. Low vertical-resolution raster displays (say, 256 pixels or less) nearly always have a fast refresh rate, usually 60 Hz, which eliminates flicker totally. For higher vertical resolution, Foley and Van Dam describe the situation [FOLE82, p. 488]:

> Many raster displays (as well as commercial TV) use an *interlaced* scan: all odd-numbered scan lines are displayed in 1/60 of a second, and all even-numbered scan lines are displayed in the next 1/60 of a second, for an overall frame time of 1/30 of a second. This interlacing helps to avoid flicker because adjacent scan lines are displayed 1/60 second apart, even though each scan line is only displayed each 1/30 second. Flicker will occur for horizontal lines one pixel wide since they are refreshed only at 30 Hz.

Unfortunately, *numerous* horizontal lines one pixel wide, and long ones at that, are the one thing almost guaranteed to appear in any segment of music: the staff! Foley and Van Dam continue:

> . . . many modern systems . . . double the rates, halving the frame time to 1/60 second. This removes all vestiges of flicker, but at the cost of more (and/or faster) refresh buffer accesses per second and significantly higher-bandwidth deflection amplifiers. An alternative is of course phosphors with a longer decay time, but then afterimages become a concern.

3.3.6. Representation and Hardware

A classic survey of hidden-surface algorithms [SUTH74] introduces the terms *image space* and *object space* for two fundamentally different internal representations of graphic data. The terms are not widely used outside of hidden-surface algorithms, but they will be useful to us. In image space, the *desired 2-dimensional picture is represented* by a 2-dimensional array with resolution equal to that with which it is eventually to be displayed; in other words, the representation is a bitmap, and its only primitive is the pixel. Thus, the representation is of the desired image and, in fact, is isomorphic to it in raster-scan form. In object space, the *objects that may appear in the picture are represented* with any resolution adequate for the ultimate display — in practice, usually with much higher resolution than the display. The primitives for a hidden-surface program are typically polygons (the faces of the objects in the scene); for music notation, they might be notes, rests, clefs, text, etc. Thus, the representation is on a higher level. It describes, not the image to be displayed, but the scene of which it is an image. For most purposes, algorithms using image-space representations are preferred because they are more efficient. However, they have the disadvantage of being resolution-dependent and therefore one must know in advance the resolution of the display desired. Furthermore, they are quite impractical for high-quality random-scan devices. There are two reasons for this. First, image-space representations obviously require execution time proportional to the number of pixels in the image, and in computer graphics high quality implies high resolution, i.e., a large number of pixels. A computation speed that is acceptable for a 512×512 CRT display is likely to be out of the question for a 15000×15000 page on a mechanical plotter. Second, image-space data is inherently in raster form, which is generally quite difficult to convert to a vector form that will really *look* equivalent on most vector devices.[17] So, if a program is to drive all kinds of graphic devices, object-space representation is virtually a necessity. (In spite of all this, it turns out that, for music setting, some type of *auxiliary* image space representation is also very desirable; see Sec. 5.2.1.)

3.4. A BRIEF HISTORICAL SURVEY OF COMPUTER MUSIC SETTING

The following historical survey of computer music setting is based on one that appeared in a 1974 paper of mine [BYRD74]. In that paper, I classified graphic output devices in the same way as I did above: into those having music characters in hardware and those not having them in hardware. I also classified systems in an obvious way as basically either "batch" or "interactive". I think this classification is still useful. In fact, it might be argued that the major change in music setting systems between 1974 and now (1983) is the same as the major change in natural language text systems in the same period: an overwhelming preponderance of batch processing has been replaced by a preponderance of interaction, with hybrid systems (sometimes called "integrated editor/formatters") now being very important.

David Gomberg has made some interesting philosophical comments on the issue of batch versus interactive computer music setting. In his dissertation [GOMB75] and a paper based on it [GOMB77], he describes the design of an ambitious and highly automated batch system and criticizes others for relying on interaction. The following quotation [GOMB75, pp. 33 – 34] makes his position clear.

> The eventual goal of this project [Gomberg's] is to take people out of the process to the fullest degree possible . . . It is expected that for certain complex situations, a

> small residue of autographic work will be necessary, but that most compositions of ordinary difficulty will require no hand finishing. [Leland] Smith takes a different view and interlaces human and computer activities. Permitting explicit interaction has allowed Smith to produce first level results quickly, and some of what he has accomplished is quite encouraging. Nonetheless, by not excluding human activity from the start, he will likely find it difficult to do so later. There is a very basic reason for removing human activities from the system, namely that these activities require knowledge and skill in many ways comparable to those of an engraver or autographer . . .

Gomberg also gives a brief survey of early music setting systems.

A much more recent and extensive but much more general survey paper is relevant to this discussion: "Document Formatting Systems" by Furuta et al [FURU82]. It is concerned with the preparation of documents of all kinds, which (at present) means primarily text, with extensions to mathematics, tables, and line drawings, although it touches on other domains, including music notation.

I shall now begin the survey. The reader should bear in mind that these systems have widely differing aims: some were really intended as music editing, not setting, systems, in particular "Mockingbird" and Kornfeld's system. See Sec. 5.3.

The first work on computer setting of music I know of was done by Lejaren Hiller and his associates [HILL65]. They were working on this problem at least as early as 1961, using the ILLIAC I computer with a modified Musicwriter music typewriter for both input and output. The main modification to the music typewriter, other than a digital interface, was installing motors and circuitry to provide discrete control of the vertical axis. Hiller's was essentially a batch system with the unusual feature of CMN input. It produced high-quality single-voice-per-staff scores, but was totally dependent on one-of-a-kind devices both for computing (the ILLIAC I) and for input/output (the modified music typewriter). Far more important, however, its software hardly went beyond simple, low-level control of the hardware capabilities of the music typewriter to automate any part of Aspects 1 and 2 of the music settting process. In fact, this system should be regarded mostly as a batch editor and hardly as a formatter at all: [HILL65] emphasizes its value for making corrections without having to redo portions of the music that are unchanged.

Harry Lincoln's batch program [LINC70] used DARMS input and drove a standard impact line printer equipped (in the manner discussed in the last section) with a print chain having music characters. It produced rather low-quality output and was quite limited in the complexity of music it could handle. With such hardware, as I have suggested, this is hardly surprising.

To my knowledge the first music setting system that did any real formatting, and the first using widely available hardware, was that of A. James Gabura [CALC67].[18] By 1967 his batch system MUPLOT was running on an IBM scientific computer with alphanumeric input and driving a standard mechanical plotter. It did single-voice-per-staff scores of high quality but with rather severe limitations.

Another early batch system was that of Norbert Böker-Heil [BOKE72], which took input in the ALMA language and again used a mechanical plotter. It could do single-voice-per-staff scores of somewhat lesser quality than Gabura's. This was perhaps the first batch system whose capabilities approached the point of practicality.

I have already mentioned David Gomberg's proposed batch system for music setting [GOMB75, GOMB77]. He planned to have input in the DARMS language. The results of Gomberg's work are 100-odd pages of narrative text and no CMN output, so his work must be evaluated very differently from others', most of which have produced very little text but substantial CMN. Gomberg describes in considerable detail, with data structures, a method for punctuation (see Sec. 4.5) in rhythmically very complex contexts, including nested groupets; he gives an algorithm for determining beam placement and slope that is probably better than any given in any book on notation (see Sec. 4.2.1); and he proposes a data structure for the overall internal representation of a piece.

Gary Nelson did some interesting work on music setting in the early and middle 1970s. His first system, NOTATE1, was written at Purdue University; it did one-voice-per-staff scores in a notation that differs from CMN in employing spatial notation of time (see Sec. 2.3.3.4). NOTATE1 produced output on an electrostatic plotter, on which it was highly dependent. More recently, at Oberlin College, Nelson has developed an integrated system of music programs called MPL, written in APL and running on a Xerox Sigma 9 computer [NELS77]. MPL's NOTATE function produces fairly high-quality scores, apparently with one voice per staff, on a pen plotter or graphics terminal.

There is one other dissertation on computer music setting, that of Charles Render [REND81]. Render's field was music education. Unlike Gomberg, Render did indeed implement a system, named MUSCOR III, but it is amazingly hard to find out from his dissertation — most of which is really a user's manual — what it actually does. On p. 7, Render makes a statement that could be interpreted either as a claim that his system can draw nearly all of the musical symbols in [ROSS70] — i.e., that it handles Aspect 3 for all these symbols — or as a claim that his system also includes virtually *everything* of any sort in [ROSS70] — that it handles Aspect 2 as well. The former claim is fairly impressive. The latter claim, if that is what he means, is very hard to credit, both because no one else has come close to doing so and because his dissertation gives almost no evidence of it, either in implementation details or samples of output (the only CMN in the dissertation is in very simple examples, and it is not even clear whether these were set by MUSCOR III). Render uses a very clumsy-looking alphanumeric encoding language, apparently of his own design, for input. MUSCOR III includes an editor. Again, it is hard to be sure, but it seems to operate on the encoding language and in a batch mode only.

Last among batch systems, my own SMUT will be discussed in detail later.

I turn now to interactive systems. One of the best-known and best music-setting systems of any kind is Leland Smith's MSS [SMIT73, SMIT78], developed at Stanford University and long in use at the Center for Computer Research in Music and Acoustics there. MSS appears to do quite well with simple to moderately complex music in a batch mode. However, its real glory is its ability, with human intervention from a graphics terminal, to do just about anything, CMN or otherwise.[19] Input is with an alphanumeric language of Smith's design and MSS can give feedback in CMN on a display screen. MSS can do scores with multiple voices on a staff, although it seems usually to require manual intervention for correct stem direction, slur position and curvature, and so on; however, it is sufficiently interactive that this can be done rather easily and quickly. It can print groupets, slurs, chords, dynamics, octave signs, and a wide variety of articulation marks and ornaments. It has a good automatic beaming capability. When its interactive graphics capabilities are used, MSS seems to be fairly machine-dependent[20]. The visual quality of MSS's notation is very good, although still noticeably inferior to engraving, for example on text, accent marks, and slurs. (As I have previously observed (Sec. 2.4), slurs are among the most difficult symbols in music notation.) It also makes stems too short under some circumstances. It produces output via electrostatic or pen plotters. Smith's system is the only one I know of that does page layout automatically in such a way as to guarantee a full last page (see Sec. 2.3.6.4). In fact, its layout facilities are exceptionally sophisticated in that they calculate how much space is needed for every measure in an entire movement before casting off, i.e., deciding system breaks. This allows MSS to make much better decisions than the usual one-system-at-a-time method can in placing system breaks as well as page breaks. (See Secs. 4.6 and 4.7.) Notation produced by MSS has been published fairly widely, for example in [MOOR77] and [CHAF82], as well as in numerous publications of Smith's San Andreas Press. One of the latter is [SMIT79], which calls itself "probably the first book on music ever to be typeset by completely computerized means". (A 1970 book referred to on p. A33 of [ROSS70] was probably not completely computer set.)

Armando Dal Molin is certainly one of the important pioneers in computerizing CMN setting. He has described his work in two papers [DALM75, DALM78]. Dal Molin has been in the music-setting business for many years, and has been working on automating the process since the 1940's, when he developed (and sold) a music typewriter of exceptional sophistication. In the early 1950's, he modified it to operate electrically and added a paper-tape punch "to provide digitized spacing and line justification". I have already described (in Sec. 3.2.6) his pseudo-clavier keyboard and "PCS" input language, which he has used in two input devices of his design. The first is the "PCS-300 Music Keypunch", a descendent of his music typewriter, with paper-tape output to send data to a minicomputer for hard copy. The PCS-300 is now obsolete and has been replaced by the "Musicomp" (formerly called "PCS-500"), a microprocessor-based graphic CRT terminal. Both devices give the operator feedback in CMN as she or he enters the music, and the Musicomp has good editing capabilities. Dal Molin produces final output of essentially engraving quality on an old photomechanical typesetter driven by the minicomputer. Beams, slurs, hairpin dynamics, etc., are all available on the Musicomp, but are currently added by hand to phototypesetter output, though Dal Molin expects to replace his phototypesetter with a new printer of some kind that will be able to print them [DALM82]. According to Dal Molin, as of 1978, over 1000 pages of music per month were being produced with his system.

A recent arrival on the music-setting scene is William A. Watkins of MUSIGRAPH. Watkins is in the commercial music-setting business, using his own system [WATK82], which runs on Radio Shack TRS-80 personal computers, interactively handles very complex textures, and produces graphically excellent results. It seems quite remarkable at first that it is practical to compute high-quality CMN on such a small and slow computer (the processor is a Z80). Not to gainsay Watkins's achievement, it is somewhat less impressive when one considers that he produces hard copy on a (photomechanical) typesetter, which is less demanding than standard graphics devices for two reasons (both of which were mentioned in Sec. 3.3.4): the computer does not draw symbols itself — the low-level work is done by the phototypesetter; and, in Watkins' words, "curved and slanted lines" (all slurs and, presumably, most beams, hairpin dynamics, etc.) are not done by machine at all, but are added manually. Regarding input of the music, Watkins says "Our system is similar in many ways to a modern word processing system in that the operator, working from a manuscript, 'types' the music which is displayed in music notation on a CRT."

The next few systems should perhaps be considered to occupy a category of their own, which might be described as "clavier-based" formatters. Most have rather limited editing capabilities.

Perhaps the first workers on input from a directly-connected instrument were Prentiss Knowlton and his colleagues at the University of Utah [KNOW71, KNOW72]. They connected a complete electronic organ, not just a keyboard, to a PDP-8 equipped with a random-scan display. In fact, what they built may well have been the first integrated computer music system that could accept input either in an alphanumeric language or from a clavier and that could produce output either in CMN or via sound synthesis (in this case analog, not digital, from the organ). Its notation capabilities appear to have been rather limited both in extent and in quality. One reason for the low quality was that "music symbols [were] constructed from a small number of lines to minimize flicker on a refreshing [i.e., random-scan] display." The system could, however, display music in a piano-roll-like notation as well as in CMN.

In Denmark, a consulting firm named Dataland, under the leadership of Mogens Kjær, has developed a very impressive formatting system called "Scan-Note". The only published information on Scan-Note I know of is rather sketchy [CMJ79]:

> The operator eaters pitches through a traditional organ-type keyboard while keying in supplementary information (duration, legato/staccato, start of slur, etc.) on an auxiliary typewriter-style keyboard. An additional typewriter is used to input text, and to further specify the final appearance of individual works.

This input is processed first by a microprocessor, then by a FORTRAN program running on a large computer. The music is finally drawn on a pen plotter "at approximately twice the desired final size. Since the Scan-Note does not accommodate every conceivable notation symbol used today, some symbols are added by hand during final editing." [CMJ79] includes a page of an orchestral score produced with the Scan-Note system. It shows most of the basics of CMN — slurs, articulation marks, dynamics, etc. It involves voice grouping (see Sec. 2.3.6.2.4), staves with two rhythmically independent voices and instances of two-note chords, all in moderately complex situations. No groupets occur, and the accompanying text does not mention them. The overall quality is nearly as high as typical engraving. I have also had access to some unpublished information [ANDE79], which adds the (unsurprising) datum that Scan-Note does automatic transposition. Dataland originally offered Scan-Note only in the form of a service, but has recently attempted to sell packaged systems to music publishers.

New England Digital Corporation has recently made available a music-setting option for their "Synclavier", one of the best-known digital synthesizers [NEDC82, NEDC83b]. It can utilize either the Synclavier's keyboard in real time (requiring the user to play along with a click track) or an alphanumeric language for input. CMN feedback and editing are done with a medium-resolution graphics terminal and hard copy is produced on a small dot-matrix printer. The editing capabilities appear in practice to be quite limited. The system can produce scores with one or two voices per staff, where each voice can include chords. When notation is done from keyboard input, groupets are apparently never generated automatically, but they can be edited in. The same is true of dynamic markings, while slurs are not available at all. Visual quality is adequate for readability, though far from engraving quality.

A rather similar system was the "McLeyvier" [CMJ82, GILB82], so called after its inventor, David McLey. It was intended as a packaged commercial product consisting of a computer, a digitally-controlled analog synthesizer with clavier, a medium-resolution (512x512) graphics terminal, and a small plotter or dot-matrix printer. The McLeyvier was unique in that from the beginning it was intended more as a tool for composers than performers, and this significantly affected software developed for it. As I have mentioned before, the McLeyvier supported non-real-time clavier input in addition to alphanumeric and real-time clavier input. Its notational capabilities were similar to the Synclavier's, and were intended to work largely automatically: only a small amount of editing could be done on its notation. The McLeyvier was briefly available for purchase, but is now being redesigned under new auspices (Syntronics of Toronto) with an all-digital synthesizer and many improvements, including much more powerful notation editing [SPIE83a, SPIE83b]. It will also have a new name, not yet decided on.

The "Mockingbird" system [MAXW82, MAXW83, ROAD81], recently developed at Xerox Palo Alto Research Center by John Maxwell and Severo Ornstein, is a very interesting variation on the real-time clavier-based formatter. An overview is given in the paper "Mockingbird: A Composer's Amanuensis" [MAXW83]:

> Mockingbird is a *composer's amanuensis*, a computer program designed to aid the composer with the capture, editing, and printing of musical ideas . . . Mockingbird is not a publisher's aid, although it does print music, nor is it a performer's aid, although it can play; it is strictly focused on the composer's need for a powerful scribe.
>
> Mockingbird is an interactive music *notation* editor. It knows nothing about the rhythmic, harmonic, or melodic aspects of music except inasmuch as they are represented in common music notation. To narrow the problem, we have concentrated on handling piano music. Mockingbird cannot presently handle orchestral scores or music for instruments that require their own notational devices.
>
> Mockingbird was written in 1980, and it is somewhat surprising that no one had previously built such an obviously interesting system. We believe that there are two principal reasons: first, we had at our disposal, for the first time, an unusually powerful set of hardware and software facilities with excellent graphics capabilities, and second, we made a number of key decisions . . . which allowed us to bypass some extremely difficult problems.

The paper discusses the problems this quotation refers to; one is the voice-assignment problem, discussed in Sec. 3.2.5 of the current dissertation. Mockingbird runs on the Dorado, a very high-performance personal computer developed at Xerox PARC. All of the software is written in the PARC experimental language Mesa. Mockingbird may optionally use a Yamaha CP-30 electric piano to record or play back music.

When music is played into Mockingbird, a score is initially produced (and displayed on a high-resolution raster-scan terminal) in a simple "piano-roll" spatial notation (Sec. 2.3.3.4). Very powerful commands are provided that allow interactive editing of all aspects of this score as well as converting it into CMN. The editing facilities include a menu selection mode which can also be used to enter music. Ornstein has written of Mockingbird [ORNS82]:

> [We] wish there were an easy way to get Mockingbird "out" [for use outside of Xerox]. But there isn't. It's based on a giant pyramid of experimental hardware and software both of which are extremely powerful and permitted us to take a major step forward relatively easily (less than 1 man-year's work).

An early interactive system was Donald Cantor's CRT-based music editor [CANT71]. It was based on the earlier work of W. B. Barker, whose program may have been the first music system to use display terminals. Cantor's system ran on an early minicomputer, a PDP-1 at Harvard University, and employed no less than four CRTs simultaneously. It was limited to two-voice music with each voice on a separate staff. Music was entered by the "menu selection" method (Sec. 3.2.3). The system could play back music (by generating square waves of the appropriate frequency), and was therefore similar in many ways to Knowlton's contemporary system described above. The major difference, from our perspective, was input method.

William Kornfeld of M.I.T. has developed a "semi-interactive" music-editing system for the M.I.T. LISP Machine. The LISP Machine has a fairly high-resolution (about 800x1000) bit-mapped display and "mouse" pointing device as standard equipment. This hardware, together with the LISP machine's sophisticated SMALLTALK-like multi-window user interface, automatically supports graphics programs with a significant amount of interactivity. Nothing has been published about Kornfeld's system except for two typical screen displays reproduced on the covers of a journal [KORN80]. The following information was given to me directly by Kornfeld [KORN82]. His program is intended primarily as an editor, not a formatter. It works in image space with a bit map. Input can be in any of three forms: alphanumeric, real-time clavier, or non-real-time clavier. To edit music, the user marks the region of interest on the screen, using the mouse. The principal editing operations are "insert", "delete", "transpose" (in the musical sense), and two rhythm-changing operations: "augment" (increase the durations by a constant factor) and "diminish" (decrease the durations by a constant factor).

Several very simple CMN systems are available commercially and may be useful for some purposes. These include Passport Designs' "Soundchaser Notewriter" [CMJ82], which runs on an Apple II personal computer equipped with a synthesizer and clavier, and a new standalone device made by Yamaha, the MP-1 "Miniprinter": a portable electronic clavier with synthesizer and built-in pen plotter that "prints melody lines on a 2-1/2 inch wide paper roll" [CAE83]. Both of these systems appear to be limited to single-note-at-a-time music. [CMJ82] also mentions the "Scorewriter" system for the Con Brio digital synthesizer, which appears to be more sophisticated, but I have not been able to obtain any details about it.

3.5. HISTORICAL OVERVIEW OF MY SYSTEM

The reader needs to know a few things about how I started working on this project to fully understand my approach. Before I knew anything about computers, I was a composer. Like every composer, I found myself spending many hours on the essentially mechanical and unmusical task of neatly recopying — sometimes with transposition — scores and parts of my pieces from barely legible pencil manuscripts. So it is not surprising that when I first learned something about computers in 1967 I thought, "I could program this machine to be my copyist", and within a few months set out to do so.[21]

The computer situation at Indiana University was for many years one of unlimited access to a "maxicomputer" (CDC 6600), with very mediocre timesharing (300 baud), and almost no interactive graphics. This was fairly typical of large universities until relatively recently. In any case, the mediocrity of the interactive facilities forced me to take basically a "batch-processing", i.e., as highly automated as possible, approach to everything.

One thing I quickly became aware of when I started programming was the enormous amount of time people at I.U. were wasting duplicating work already done. I saw two reasons for this: (1) programs written for one machine usually could not easily be run on another; and (2) programs performing related functions that could have been used together often had incompatible formats. These observations led me (1) to write my programs as portably as possible, and (2) to work towards an integrated system of music programs — including analysis and sound-synthesis components as well as notation — that could easily be used together. The resulting "Indiana University Computer Music System" is diagrammed in Fig. 6. Arrows represent the flow of information; names in smaller print represent interface programs. SMUT is, of course, the music-setting program whose development was the kernel of my research.

In accordance with my philosophy, and since a translator for MUSTRAN (described in Sec. 3.2.4) already existed, I decided to use MUSTRAN as an input language. By calling the program that recognizes MUSTRAN a "translator", I mean that it translates MUSTRAN from a free-format context-sensitive form relatively convenient for humans into a fixed-format context-independent form much easier for another program to handle. However, the format of the information produced by MUSTRAN is not quite suitable for input to a notation program; besides, MUSTRAN provides no high-level control features at all (see Sec. 4.3, 5.3.1, and [BYRD80] for discussions of such facilities). To translate MUSTRAN output into SMUT input and facilitate high-level control, I wrote an interface program, called SMIRK. (The other pathway shown into SMUT, through JANUS, is designed for compatibility with composing programs, such as MUSC and STOCHOS, and digital sound-synthesis programs, such as MUSIC V.)

Additional background and discussion of the implications of these basic design decisions, especially portability, is given in my paper [BYRD80]; however, I would like to make some additional comments here. SMUT's overall structure was influenced heavily by two factors: my desire to make it portable, and its batch orientation. These factors are interrelated: the desire for portability was one reason for the batch orientation. Another result of the desire for portability is SMUT's reliance on sequential I/O only, considering that until quite recently no widely available higher-level language supported direct-access I/O in a reasonably machine-independent way. This restriction to sequential I/O forced me to find ways to do the formatting that relied at each step very largely on local information. (As Gomberg's quotation in Sec. 5.4 makes clear, this is hardly the way a person formatting music manually works. Here is a case in which, it seems to me, properly imitating manual techniques would pay off; see Sec. 5.4.) The need to keep information localized, in turn, led to SMUT's four-pass structure.

Chapter 4 is devoted to a detailed description of SMUT. However, a summary of its capabilities comparable to those I have given for other systems should be helpful. SMUT produces high- but not engraving, quality scores with one or two voices per staff. Aside from integration into the IUCMS, its outstanding features center on two areas:

(1) Portability, both in terms of CPU and graphic output device. In 1980, I converted a version of SMUT from the original 60-bit maxicomputer (CDC 6000/CYBER) version to a 32-bit virtual memory superminicomputer (PRIME 50 series); in 1983, Kimball Stickney converted a later version from maxicomputer to another, similar superminicomputer (DEC's VAX under the VMS operating system). Each conversion took 20 or 30 hours of work, less than one percent of the 5000 or more hours involved in developing the original version. Also, SMUT has been used to drive numerous output devices of all types — random- and raster-scan, display and hard copy — with essentially no incremental work per new device, simply because of its reliance on extremely simple graphic primitives (see Sec. 4.7). See [BYRD80] for more discussion of portability issues.

(2) Automation. SMUT, with some aid from SMIRK, provides several important high-level user-interface features: *(a)* transposition, i.e., changing pitches by a constant; *(b)* automatic clef changes to avoid excessive ledger lines; *(c)* automatic breaking of multibar rests into series of whole-rests; *(d)* performance directions whose printing is conditional on their belonging to the top voice of a group; *(e)* automatic rhythm clarification (see Sec. 4.4.1); and *(f)* automatic beaming, beam positioning, and fractional beam pointing (see Sec. 4.4.2). Features *(a)* through *(d)* are especially useful because they make it possible, in most cases, to encode a piece of music in such a way that the user can obtain from it either a score or a set of parts, with few or no changes to the data. Doing so requires one or more of these features in most cases. Features *(b)*, *(e)*, and *(f)* should be particularly useful to persons who are employing a computer as an aid in the composing process, as suggested by (among others) Iannis Xenakis [XENA71]; in fact Xenakis' composing program STOCHOS has been interfaced to SMUT, as shown in Fig. 6.

NOTES

[1] One difficulty in putting together such a survey is that much — perhaps most — of the relevant work has been done in nonacademic settings and is not readily available. This is especially true in recent years, as computer music setting has gained considerable commercial importance. As a result, a large amount of important work is unpublished. However, I have obtained quite a bit of unpublished information from the researchers themselves, most of whom were very cooperative.

[2] [MOOR75] describes a precursor, the "Melograph" (not to be confused with the 18th-century melograph I have previously mentioned, which undoubtedly was driven by a directly-connected instrument). This was a "special-purpose piece of mostly analog hardware and a chart recording scheme" developed for the UCLA Ethnomusicology Department which, for monophonic music, could produce a graph of pitch as a function of time. Neither the 18th-century nor the 20th-century melograph produced anything like CMN.

[3] In the graphics literature, tablets are usually described as "locators", and pointing devices are usually described as "picks". In this terminology for interactive input devices, real picks are rarely used; instead they are usually simulated, most often by locators. I will generally avoid this terminology because I feel that, for our purposes, it would cause confusion more than it would help.

[4] Again, an analogy to text processing presents itself: the "on-line character recognizer". See [NEWM79, p. 202ff].

[5] The developers of DARMS argue that using a letter name for the notation of a pitch implies a low-level music-theoretic decision that should not be part of the encoding process. The justification for this claim is too technical to discuss here, although it has to do with the relationship between vertical position and pitch. The principle is certainly valid, but I find its application here totally unconvincing. Another argument in favor of the DARMS technique is given by Gomberg [GOMB75, p. 14].

[6] I say "almost certainly not as well" on user-interface grounds. It is very difficult for a system in which the user's data is run through several programs in succession to give error messages as helpful as those that can easily be given by a single program.

[7] The meaning here of the phrase "explicitly says" deserves some comment. In a typical system of the type I am describing, the keyboard controller periodically sends the computer a string of bits each of which represents a key, with "1" bits representing keys that are currently depressed. From the viewpoint of a program, this is certainly a more explicit representation of pitch than it would get from sampling the acoustic signal generated by a performance. The human viewpoint can be quite different, however. Imagine a situation where, let us say, the Duchy of Grand

Fenwick is at war with the United States and forbids the possession of recordings of the "Star Spangled Banner". Would a court be more likely to convict someone who has an ordinary audio tape of an organ performance of the piece, or someone who has a magnetic tape containing digital samples of the same performance? Very likely the former. Lest this sound too silly, a pornography trial was once held that turned on a similar question for videotape.

[8] Note, incidentally, that this entire discussion is based on the assumption that the performer is trying to play in a constant tempo (and, of course, that the computer knows what the tempo is). If this assumption is violated, as it nearly always is in ordinary performances — i.e., ones that are not for the benefit of computers — the problem is far more difficult.

[9] It may be objected that voice assignment of a note does not always have notational consequences. This is true, but it has such consequences so often that the problem cannot be ignored.

[10] The crossing of voices also causes serious purely notational (typographic) difficulties, as mentioned in Sec. 2.3.6.2.

[11] The same classification scheme is appropriate for methods of loading the memories of *digital sequencers*, devices that are used in conjunction with electronic music synthesizers to record series of events (ordinarily notes) so that they can be recalled and played back with, typically, a single keystroke. See the article by Devarahi [DEVA83]. The term "single-step", commonly applied to sequencers, is equivalent to my "non-real time". Devarahi says "In single-step loading, each note is entered as an isolated event, either from a calculator-type keyboard or from a [clavier]."

[12] A remarkable book — [CARL78] — exists which seems by implication to "give the lie" to my statements about the limitations of music typewriters. It contains amazingly complex and subtle pictures made on ordinary Roman-character typewriters. However, the book makes it clear that the pictures were done with such techniques as rotating the paper in the carriage, varying pressure on the keys (of nonelectric machines) to change contrast, doing fractional spacing, typing through cut-outs, etc. The enormous amount of work this will take for nontrivial results is obvious, so that doing complicated graphics by typewriter is really an example of what has been called, with apparent reference to the Turing machine, the "Turing Tarpit": a technique that makes everything possible but little or nothing practical. More direct evidence of the utter impracticality of doing CMN on conventional typewriters is provided by a statement by one of the artists whose work appears in [CARL78], Rochelle Vickey [VICK83], that "I have attempted several times to type out music. Of all the feats I've performed with the typewriter, that one completely defeated me!"

[13] One music publisher, Novello, seems to disagree. They have used movable type for a substantial amount of music, including such a complex work as Bach's *B-Minor Mass*, published by them in 1907. I do not know if they are still using movable type.

[14] Knuth gives another brief history of typesetting technology [KNUT79, pp. 15 – 16] in which

he argues that 1000 points per inch resolution is, for all practical purposes, perfect:

> I was quite skeptical about digital typography, until I saw an actual sample of what was done on a high quality machine and held it under a magnifying glass: It was impossible to tell that the letters were generated with a discrete raster! The reason for this is not that our eyes can't distinguish more than 1000 points per inch ... [but] that particles of *ink* can't distinguish such fine details — you can't print the edge of an ink line that zigzags 1000 times on the diagonal of a square inch, the ink will round off the edges.

[15] Among commercially available products, as of early 1983, the highest resolution random-scan displays are about 4000 by 4000; the highest resolution raster-scan ones, about 1300 by 1000. For hardcopy devices, the figures are invariably given as points over a linear distance, not total resolvable points. Random-scan hardcopy devices (and phototypesetters, as I have already said) frequently exceed 1000 points per inch, while raster-scan ones rarely exceed 300. Many of these numbers could be raised by a factor of about 2, depending on how "commercially available" is defined.

[16] Gray levels are not quite irrelevant, in that — properly used — they can make a display appear to have much higher resolution than it actually has. The technique, which is of great importance in many types of graphics, is called *antialiasing*. See [FOLE82], p. 436. Also, color played an important role in music notation at various points in history, and it is sometimes used now in printed music for didactic purposes, although neither usage could possibly be considered part of CMN. For examples of music set in color by my system, see [HOFS82a].

[17] This is because most vector devices, hardcopy or display, behave rather idiosyncratically at the endpoints of vectors. For example, vector hardcopy devices usually depend on ink which takes a measureable distance to start flowing evenly. The inverse operation — converting from vector to raster form — is straightforward and, in fact, is generally referred to simply as "scan conversion" (see [FOLE82]).

[18] [RASK80] includes a fairly high-quality sample of music, four measures in length, printed on a pen plotter by a program of Raskin's in 1967. He gives no details of the program at all; however, according to [GOMB75], Raskin

> produced a rather standard graphics package which produces arbitrary items on a plotter. By specifying coordinates, lengths, etc., he produced a drawing which happened to be four measures of music, but which might as well have been a portion of a circuit diagram. There was no attempt to determine the placement of the components in any automatic way.

Such a system attacks only Aspect 3 and, therefore, is more comparable to Hiller's music typewriter than to Gabura's system or to most of the others discussed here.

[19] In his dissertation [GOMB75], Gomberg used two pages from the *Double Concerto* by Elliott Carter as examples of great notational complexity. Smith has printed one of these pages with his system, slightly simplified, as a demonstration.

[20] This direct relationship between use of interactive graphics and machine-dependence is to be expected. See [BYRD80].

[21] My first estimate of how long it would take to have a usable program was on the order of a few months, not 14 years. Underestimates this extreme of the difficulty of programming CMN seem to be rather common.

4
How SMUT Works

4.1. INTRODUCTION

As I have pointed out (in Sec. 3.1), one can view music setting as having three aspects: selecting the symbols to print, positioning them, and actually printing them. My system, SMUT, is concerned with all three. It has a number of automatic features that affect what symbols it prints. Nearly all of these can be disabled, but a few (related to system breaks: see Sec. 4.5) cannot be, since I feel that what they do is so deeply embedded in the syntax of CMN that it would not be worth the effort of making them selectable.

Section 2.3 described the basic logic of CMN, intended, as I said there, to impart a "reading knowledge" of music notation. It did not give any of the rules for selecting and positioning symbols that are part of "writing knowledge" — rules whose discovery and implementation has been a major part of my work. As I said in Chapter 1, "The closest thing I know of to an explicit statement of the rules of music notation . . . is still no more than, to choose a number, 5% explicit (as compared to the usual value of, say, 2%)." In this chapter I will first try to give the reader some intuition about the problem by pointing out some questions that arise in a fragment of simple single-voice music, in a complete page of score by Bach with one voice per staff, and in a fragment of music with two voices per staff. As I proceed I will compare these rules to those given by other writers. I will next give an overview of the organization and operation of SMUT, and then describe some of its more interesting features in detail. I will conclude the chapter by "talking through" the beginning of the same page of Bach, giving much more detail.

Note: in this chapter I will criticize my work as I go, using the convention of enclosing self-critical comments in bold curly brackets, viz., "**{** . . . **}**".

4.2. SOME BASIC QUESTIONS OF POSITIONING AND SPACING

4.2.1. One Voice

Fig. 1 is a short fragment of a Stephen Foster song containing a few quarter- and eighth-notes, with no accidentals or any other complicating symbols. It already brings up a number of questions, however. Should note stems go up or down? How long should they be? These questions seem at first to have very straightforward answers, and, in fact, they do — in sufficiently simple cases. Ross says [ROSS70]:

> A note on or below the second space [from the bottom of the staff] has an up-stem on the right side of its notehead. A note on or above the middle staff line has a down-stem on the left side of its notehead.

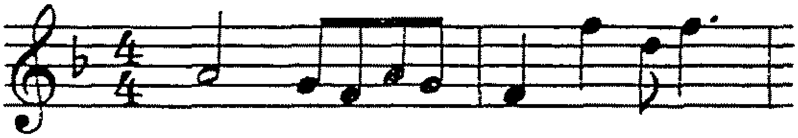
As for length, Stone says [STON80]:

> stems on single notes should be one octave long unless the note is farther than one octave from the middle line of the staff, in which case the stem is lengthened to reach the middle line.

The standard texts, and SMUT, all agree on these simple rules for simple cases.[1] What constitutes a non-simple case, then? Stone suggests one: non-single notes, i.e., more than one notehead on a single stem. Another case is when two voices share a staff; the rules for this case are given in Sec. 2.3.6.2.1. Neither of these situations occurs in our little fragment, but a third one does: notes in beamed groups with stems. Here, each stem must end at the beam, a constraint that will nearly always be inconsistent with the individual notes' normal requirements for stems, in terms of direction, length, or both. The rules here are fairly well agreed on but far from simple. Ross devotes no less than 20 pages to this question, mostly in tables. Gomberg [GOMB75] expends less paper on the point than Ross, but probably sheds more light on it than Ross or anyone else has; in any case I will not enter the fray here. I will, however, say something about SMUT's method in Sec. 4.4.2.

A question of a different sort is, how much horizontal space should be inserted between symbols? The principle was expounded in Sec. 2.2.3.3, namely that the ideal distance between consecutive time-occupying symbols — notes or rests — increases with time, but more slowly than linearly. The nonobvious formula actually used by SMUT, involving exponentiation and logarithms, is given

Figure 1. Positioning and spacing: one voice



in Sec. 4.6. This ideal spacing is actually realized in Fig. 1, because of its simplicity, while it is violated continually in real music, mostly for those short notes that need more than the ideal amount of space. This might be because of accidentals or dots on the short notes, or it might simply be because the (human!) editor did not want to allocate more than the minimum possible space to, say, 500 32nd-notes in a piece so that a very few 64th-notes could be closer together than the 32nds. See the next section and Chapter 2, Fig. 32.

4.2.2. Several Staves, One Voice Per Staff

A page of music by Bach, the first page of the so-called "Sloth Canon" from the *Musical Offering*, BWV 1079, is given as printed by SMUT in Fig. 2. Several points of interest are labelled with numbers (enclosed in circles); I will now comment on them, one by one.

(1) The *heads* of notes occurring at the same time are (when possible; see Sec. 2.3.6.2.1) aligned, not the stems or the first components of the note "hunks" (Sec. 2.3.6.2.1), which include accidentals before the head and flags and dots after it.

(2) Flags go to the right, whether they are above or below the notehead.

(3) When a slur[2] extends across a system break, it is replaced with two slurs, one at the end of the first system and one at the beginning of the second. Notice that the left end of the slur that crosses into the beginning of a system cannot have a fixed left x-coordinate because it cannot begin until after the key signature, whose width can vary greatly (down to zero).

(4) See item (5).

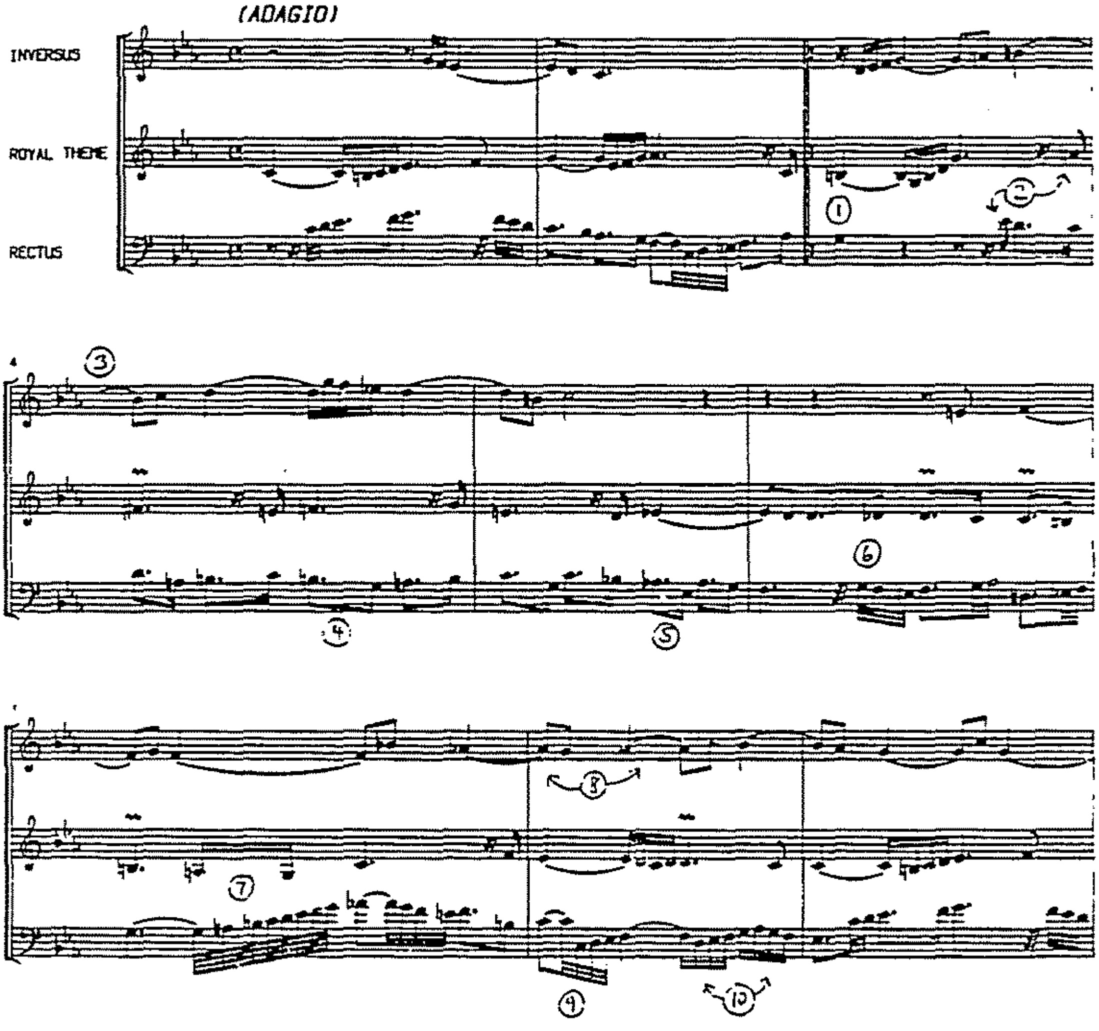
(5) This dotted-eighth-plus-16th group gets its ideal space allocation, since nothing in this or the other voices disturbs it. The similar group at (4), on the other hand, gets considerably more space in order to accomodate the top voice. This clearly shows the violation of ideal spacing I discussed in the previous section (Sec. 4.2.1).

(6) Alignment here is interesting. The D-flat, the fourth note of the measure in the middle voice, is positioned further to the right than it would ideally be in order to leave room for the lowest voice. The 32nd-rest in the lowest voice starts in the middle of the middle voice's dotted eighth, and it is

SLOTH CANON

FROM THE MUSICAL OFFERING, BWV 1079

J.S. BACH
(1685-1750)



SMUT 2.0 SAMPLE PLOT FROM MUSTRAN DATA
DONALD BYRD, WCC, INDIANA UNIV., BLOOMINGTON, IN 47405

Figure 2. Positioning and spacing: several staves, one voice per staff in the Bach *Sloth Canon*, score (set by SMUT).

positioned accordingly. Notice, however, that no additional space was needed for the flat sign in the middle voice, which overlaps the second notehead in the lowest voice. This poses no problem for the reader of the music.

(7) This is another, rather dramatic, demonstration of non-ideal spacing. The 32nd-notes in the bottom voice are spaced very nonuniformly because of the naturals attached to two of them. As a result of the demands of the bottom voice, augmented by the two naturals, the 16th-notes in the middle voice are much further apart than they "should" be.

(8) Compare the two ties, which involve notes of identical pitch and even duration: one is above the notes while the other is below. This is a consequence of the differing stem directions of the eighth-notes in the two situations, which difference results in turn from the beamed groups that the eighths are involved in. The first beam requires both of its notes to have stems up (see Sec. 4.4.2), which is normal for the tied notes, since they are in the second space; but the second requires both of its notes to have stems down[3], and the usual rule for slur placement is that the slur goes above the notes if one or more of the notes it encompasses has a down stem.[4] Also, the left end of the second tie had to be moved right from its normal position in order to avoid intersecting the note's stem.

(9) The stems of several of the notes in this beamed group have been extended considerably so that none will be too short.

(10) These two beamed groups are very similar-looking, but their overall pitch contours are reflected by the beams' slopes: the one on the first is horizontal while that on the second tilts slightly down.

Two global decisions are: where system breaks should go; and what groups of notes should be beamed together. The former is in priuciple very intuitive. The main principle underlying the latter is to show the underlying metric structure of the music as clearly as possible.

4.2.3. Two Voices Per Staff

Chapter 2, Fig. 48 (Beethoven, *Symphony No. 9*, III, p. 138, clarinets) is another fragment, but a much more complex one than Fig. 1. It illustrates some of the problems that arise in music that has two voices per staff, many of which were mentioned in Sec. 2.3.6.2. Note stems must all be pointed up for the upper voice and down for the lower; slurs, ornaments, fingerings, groupet accessory numerals, etc., must be placed towards the outside of the staff. By far the most difficult, hunks need to be moved or even internally modified when either their noteheads, accidentals, or dots collide: these occur in the second, third, and fourth measures of the figure. Doing this automatically is beyond SMUT; it knows nothing of inter-voice collision problems. See the next section and Sec. 5.2.1.5.

4.3. PRINCIPLES OF SMUT'S OPERATION

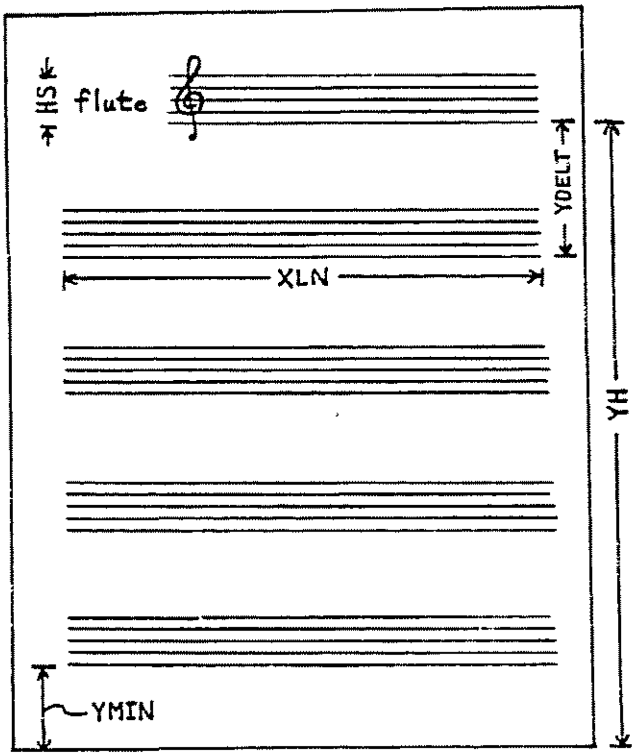
Internally, SMUT consists of an initialization phase plus four "passes", which roughly move down the list of aspects given above, progressing from the abstract towards the concrete. One important way in which this is visible is in the coordinate systems used, which I will describe when appropriate.

In the **initialization phase**, one "layout line" and several "initialization lines" are read. The layout line describes the page layout and also gives various global parameters for the run, such as whether the run is "normal" or "edit" mode. In normal mode, all four passes are run. The most basic option is whether a score or a set of parts is to be produced. Most of the problems involved are identical.[5] In edit mode, there are two options: either the first three passes can be run to produce intermediate files describing a score or set of parts; or only pass IV can be run (with intermediate files from a previous run) to actually print the score or parts. These options make it possible to run separate programs between passes III and IV for special purposes such as interactive graphic editing or combining music with separately formatted text.[6]

The page layout is specified by five parameters, as shown in Fig. 3: height of staff (HS), length of staff (XLN), y-coordinate of the bottom of the top staff (YH), minimum y-coordinate for the bottom of the bottom staff (YMIN), and vertical distance between staves (YDELT).

Most of the initialization lines control features similar to those in typical natural language text formatters: titles, running footers, etc. Two, however, are specific to music formatting. One controls "grouping of voices", a specification of a hierarchy of the type described in Sec. 2.3.6.2.3, including which voices (if any)

Figure 3. SMUT page layout parameters



share staves.[7] Voice grouping determines whether barlines extend across staves — they join "related" voices into groups and are broken between groups — and how conditional performance directions are handled (see Sec. 3.5 and Fig. 14). Voices sharing a staff have their stems pointed away from each other, slurs, ornaments, and groupet accessory numerals put on the outside, and so on. {Problem: SMUT is totally ignorant of the danger of collisions between two voices sharing a staff. In particular, when their notes are a second apart or in unison, the heads will overlap or be superimposed (see Sec. 2.3.6.2.1), and when both have accidentals and are within a sixth, the accidentals may overlap. See Sec. 5.2.1 for a lengthy discussion of such *collisions*.} The other music-specific command specifies how many measures are to go in each system of music; if present it clears flag SYST to disable automatic placement of system breaks (see below).

Pass I is primarily the symbol-selecting pass. It reads the entire data file, which must be arranged as all of voice 1, then all of voice 2, etc. Depending on the options enabled by control commands, it performs various "high-level" functions, as follows:

(1) Deciding what accidentals to use, both regular and cautionary, if the flag CHROM is on. (*Cautionary accidentals* are accidentals that are technically redundant but are included as reminders to the performer; they are often printed in parentheses. For example, if C-sharp appears in the key signature and one measure has a C-natural, a C in the next measure without an accidental may have a sharp in parentheses in front of it. See Sec. 2.5.) {Problem: it is not always possible to determine cautionary accidentals for voices sharing a staff by examining the voices independently. For example, if a sharp appears on an F in one voice on a staff, a following F-natural in the other voice should always have an explicit natural. See [STON80], p. 166.} SMUT allows pitch input in two forms, selected by CHROM. If CHROM is on, pitch is described "chromatically": a statement of the actual pitch is made, not of the notation, and SMUT chooses the appropriate enharmonic representation. The CHROM pitch descriptor is simply an integer giving the number of semitones above the bottom note on the piano, A0 in ASA notation (Sec. 2.3.2). So C4 — middle C — is 39. A note with pitch descriptor

40, then, might be written as a C with a sharp in front of it, a D with a flat in front of it, or (depending on the key signature and previous accidentals in the measure and assuming that no cautionary accidental is needed) as just a C or D with no accidental. If CHROM is off (the usual choice), SMUT expects an explicit description of the notation.

(2) Adding barlines, under control of the flag BAR. If BAR is on, SMUT sums the durations of notes and rests as it goes, continuously compares the elapsed duration in the current measure to the proper measure duration (as specified by the current time signature), and generates barlines when needed.

(3) Rhythmically "clarifying" notes and rests, if the flag DESYNC (short for "desyncopation") is on. That is, any rest that makes the rhythm hard to read by obscuring the beat is broken into a series of consecutive rests, and any note that makes the rhythm hard to read is broken into a series of consecutive tied notes (which I will call "subnotes"). The algorithm for this will be discussed in detail in Sec. 4.4.1.[8] If a note that has been broken down had a glissando on it, SMUT generates appropriate intermediate pitches for the subnotes; otherwise all of the subnotes have the same pitch. Note-related symbols are moved to the appropriate subnote (usually the first).

(4) Deciding what notes to beam together, under the control of the time signature and the flag BEAM, and which way tc point fractional beams (with algorithms discussed in Sec. 4.4.2).

(5) Generating whole rests (separated by barlines, of course) from multibar rests if SMUT is making a score. (Handling of multibar rests is the only situation in which Pass I cares whether it is doing a score or a set of parts.)

(6) Deciding when to change clef, if specified by the flag CLEF. When the user sets CLEF, she or he must specify which clefs the program may choose from; this is necessary since, as pointed out in Sec. 2.3.2, CMN employs only a subset of the existing clefs for any given instrument. Of course, changing clefs affects the vertical position of all notes, as well as of many other symbols (slurs, groupet accessory numerals, etc.), following. A good algorithm for deciding when to change clefs would be fairly complex, requiring, at the least, some lookahead. A really good algorithm would also need considerable knowledge of the semantics of music; see Sec. 2.5. SMUT is very simpleminded about this, however: with some minor restrictions, it simply changes

clef whenever it encounters a note that would need more than four ledger lines in the current clef and changing to one of the allowed clefs would reduce the number of ledger lines.

(7) Handling SMUT's very simple control mechanisms, namely loops with fixed iteration counts and conditionals.

Pass I also makes some positioning decisions, such as: whether beams go above or below notes, and approximately where; whether slurs go above or below notes; groupet accessory numeral vertical positions; and stem directions (up or down) and stem lengths. (For beamed notes, stem lengths assigned here are approximate, since they depend on the position of the beam, which cannot be decided exactly until Pass III: see Sec. 4.6.) All of these decisions are affected by the SHARE flag. This flag is automatically set at the beginning of each voice to indicate whether the voice is the upper of a shared staff, the lower of a shared staff, or has a staff all to itself, but the user can (for special purposes) explicitly set it to any of these values at any time. Pass I writes its output on scratch file 1, still with all the data for each individual voice together. The coordinate system used here is, for the time/x axis, "attack time" (see Sec. 4.6); for the pitch/y axis, an integer number of "half-lines" relative to the bottom line of the staff (so, for example, a note on the top line of the staff has y-coordinate 8).

If a score is requested, **Pass II** sorts the Pass I output file by measure onto scratch file 2. In order to align everything properly in the score, SMUT needs to examine everything happening in measure 1 across all voices, then everything in measure 2 across all voices, etc. {Problem: This method will not work for the significant body of music in which barlines do not always coincide, i.e., measures do not always begin or end together. However, this cannot occur in CMN, and being able to slice the music into independent sections considerably simplifies matters. See [GOMB75] for an approach to music setting that does not assume coinciding barlines.} Of course, this is much more easily done if the input is sorted by measure, which is the *raison d'être* of Pass II. If a set of parts is requested, Pass II does nothing except output some statistics.

Pass III is concerned mostly with positioning. It reads scratch file 2, one bar of score or of a part at a time, and assigns final y-coordinates to several symbols that could not be fully handled by Pass I — for example, beams. More importantly, in its "justification" subroutine JUSTIF, Pass III uses the time information it receives to assign temporary (because right-justification has not yet been done) x-coordinates to all symbols, i.e., it does *global punctuation* If SYST is on, it decides as it goes when the current system is full and a new one must be begun. The process is quite complex; it is described in detail in Sec. 4.5. Pass III writes records, each of which describes one musical symbol, onto scratch file 3, and control information — x- and y-position offsets for the system, the number of records for the system on file 3, etc. — onto scratch file 4. The coordinate system used now has floating-point position values for both x and y, normalized for a staff of height 1 with its lower left-hand corner at (0,0).

Pass IV reads scratch files 3 and 4 and actually does all of the printing. As it goes, it multiplies the temporary x-coordinate of each symbol by an appropriate constant so that the system is stretched to just the right length (as given on the layout line), unless right-justification has been turned off either globally or for this system alone. It further scales y-coordinates by the actual height of the staff desired, and offsets y-coordinates by the position of the bottom of the staff, to produce coordinates in the graphic output device's coordinate system. Pass IV achieves device independence by using only a very few vector-drawing primitives. See Sec. 4.7 for more details.

4.4. PASS I: SINGLE-VOICE PROCESSING

I will concentrate here on Pass I's two fancy features relating to rhythm treatment. N.B. These two sections involve many technicalities of a musical nature that are not important to an understanding of the remainder of this dissertation. They do, however, go to show how much nonobvious complexity exists in music notation. The reader may wish simply to look at the examples to see what I am driving at.

4.4.1. Rhythm Understanding: Clarification to Insure Readability

The difference between technically correct rhythm notation and readable rhythm notation is not well known, although most musicians know that there *is* a difference. One important aspect of rhythm notation has to do with not "obscuring the beat", that is, making sure that the underlying rhythmic structure of the meter is always recognizable. This requires in some cases writing what could be a single rest or note instead as two or more rests or tied notes: see Fig. 4. Curiously, the notation books make only a few perfunctory comments on this subject.[9] The clearest statement of the principles I have seen is in an elementary theory text [WINO79, p. 114]:

> There are two guiding principles for all rhythmic notation: 1. The notation should show as clearly as necessary the underlying metric organization. 2. The notation should be as concise as possible.

While perfectly correct, this statement is far from being explicit enough for implementation on a computer (and, to my knowledge, nothing more definite has ever been published). [WINO79] goes on to describe several ways to notate a single rhythm, ranging from abysmal to excellent; for the (human) reader, these do indeed make the idea much clearer. But — considering the present limitations of artificial intelligence — examples are not of much help to a computer. It is certainly not immediately obvious from any set of examples how the rules should be formalized. Consider the patterns to the left of the arrows in Figs. 4*a*, *b*, *d*, *e*, *f*, and *g*: these need to be rewritten as shown on the right of the respective arrows. (I will explain the numbers below the notes later.) On the other hand, Fig. 4*c*, which looks quite similar to 4*a* and *e*, is all right as it is. A first guess at the rule might be "subdivide a note or rest at the first internal point with rhythmic strength greater than the attack point." (The term "rhythmic strength" was introduced in Sec. 2.3.3.2; we will shortly give it a more formal definition.) This is certainly too strong — it fails on Fig. 4*c* and, depending on whether or not it is applied recursively, on either *b* or *f* — but is on the right track.

What constitutes clear, concise rhythmic notation? For one thing, it is obvious that no note or rest should ever extend into the beginning or across the end of a groupet: having a quarter-note partly inside a triplet and partly outside might be concise, but hardly clear. But to answer the question completely, we need to know what the normal rhythmic structure of a measure is in whatever meter we are concerned with. The discussion of meters in Sec. 2.3.3.2 gives us some of this structure, including the term "rhythmic strength"; let us continue the analysis with an example along the lines of Chapter 2, Table 1. In 4/4 time,

Figure 4. Rhythm clarification

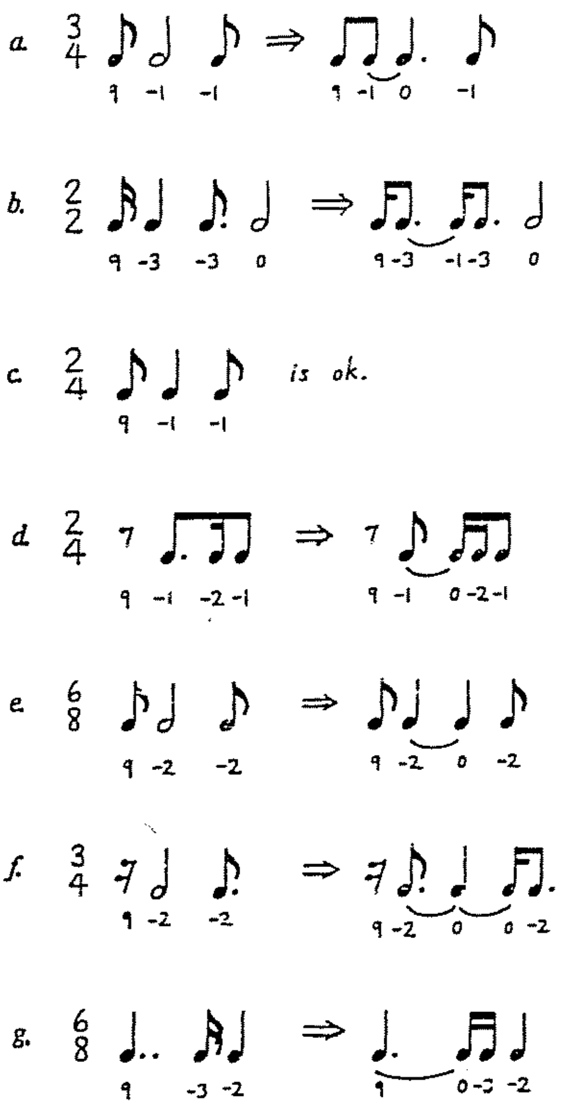
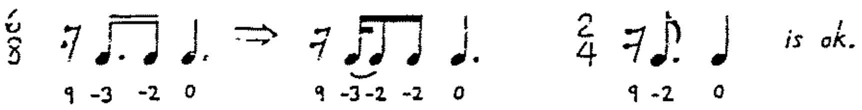
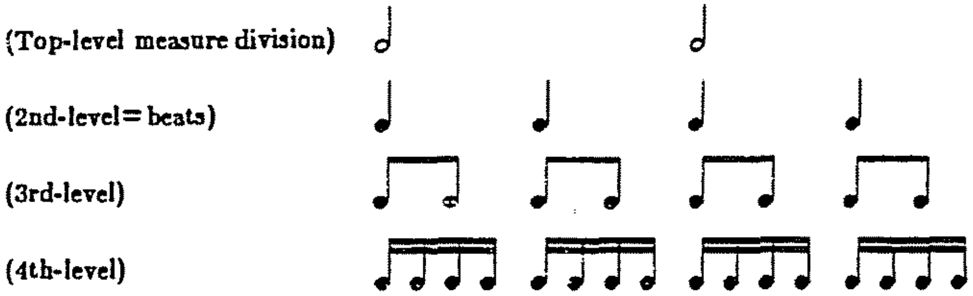


Figure 5. Compound vs. simple meter

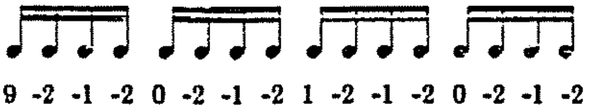




We now have an implied hierarchy of rhythmic strength below the beat, but we also need a hierarchy of beats within the measure.[10] This is not hard. Firstly, the downbeat (the first beat of the measure) is always the strongest beat by far. In meters with 3 beats per measure, the second and third are equal; with 4 per measure, the second and fourth are equal but weaker than the third. Continuing up through 8 beats per measure, we might arrive at the following table:

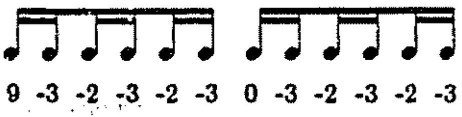
beats per measure	strength pattern (0= weakest beat)
1	9
2	9 0
3	9 0 0
4	9 0 1 0
5	9 0 1 0 0 or 9 0 0 1 0
6	9 0 0 1 0 0
7	9 0 1 0 1 0 0 or 9 0 1 0 0 1 0 or 9 0 0 1 0 1 0
8	9 0 1 0 2 0 1 0

Evidently, two-part division of each level is predominant, with three-part division used where unavoidable. The "9" used above for the downbeat is not at all sacrosanct, but could be replaced by any number larger than about 4. Less arbitrarily, I have used 0 for the weakest beat, with consecutive positive integers for successively stronger levels above it.[11] These conventions are important in the algorithm I am about to present, and can easily be extended to include the below-the-beat hierarchy we have already discussed. We can simply use consecutive negative integers starting with -1 for successively weaker levels below the beat. The complete pattern down to two levels below the beat in 4/4 time is, then:



For completeness, we need one more thing: a way of assigning rhythmic strength to temporal points within groupets that do not coincide with any time that can be expressed without groupets.[12] This would not be too hard to do along the lines we've been following. In effect, groupets set up a temporary "metric environment". However, I will oversimplify here (as SMUT does) and assign to all such points a large negative value, say -9. This will lead to incorrect results only in a few very complex situations (but see Sec. 4.4.2).

Matters are slightly more difficult in compound meter. (The rule SMUT uses for classifying meters as simple or compound was given in Section 2.3.3.2.1.) Above the level of the beat, the situation is identical, but below the beat, compound meters have two peculiarities. One is suggested by the non-parallelism of Figs. 4*c* and 4*e*. Also, consider the rewriting necessary in Fig. 5*a*, whose simple meter analog (Fig. 5*b*) requires no rewriting. A little thought shows that the difference amounts to a much stronger need in compound meter, as compared to simple meter, to divide at the level of the beat and at the first level below the beat. Note that nothing comparable shows up at any lower level; consider for example Fig. 6. So, compound meters will largely be taken care of simply by skipping -1 when assigning strengths, that is, the level below the beat will be -2. The complete pattern down to two levels below the beat in 6/8 meter is therefore



(The numbers beneath the notes in the figures are, of course, rhythmic strength labels.)

We are finally ready to discuss the rhythm-clarification algorithm. It is as follows:

(1) Divide at any point >1 stronger than the attack.

(2) If there is more than one point exactly 1 stronger than the attack, divide at the *first* such point. (These two steps correctly divide Figs. 4*a*, *b*, and *e*, but leave *c*, *d*, *f* and *g* alone.)

Figure 6. Low levels in compound meter

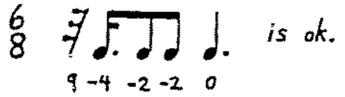


Figure 7. A duration that cannot be notated with one note

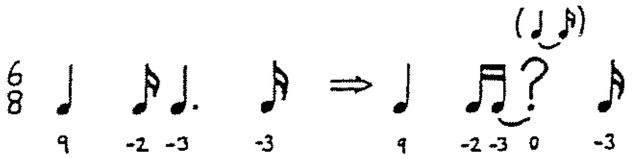
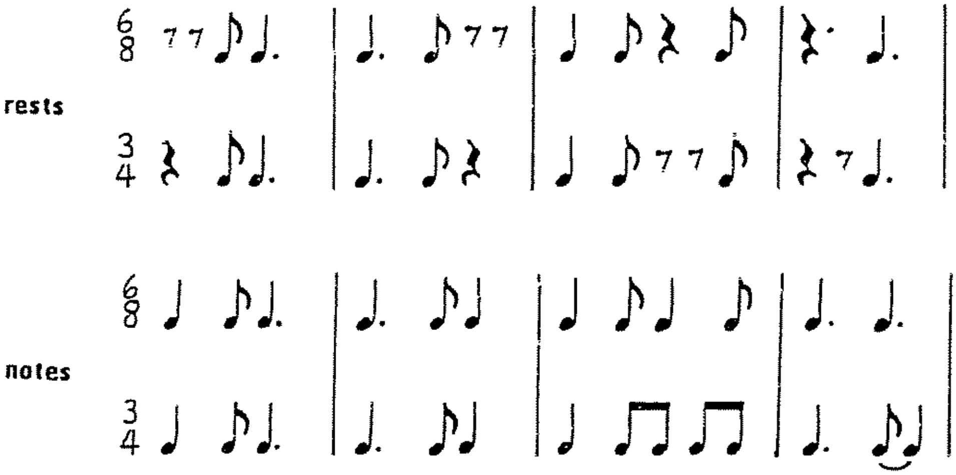
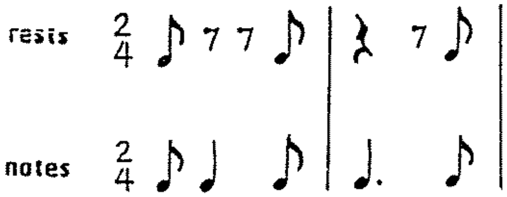


Figure 8. Proper notation of rests and notes is different.





(3) If the note or rest crosses any point stronger than the attack and ends at a point *weaker* than the attack, divide at the occurrence of higher strength (only one is possible, because of rules 1 and 2). (This takes care of Fig. 4*d*.)

(4) If the note or rest has now been divided, its first piece is now satisfactory; apply the procedure recursively to its second piece. (This takes care of Fig. 4*f*; *c* and *g* are still unaffected.)

N.B. It is not obvious that this algorithm always generates notatable durations, i.e., durations that can be represented by a single note rather than several notes tied together. (It will be recalled that possible single note durations are of the form $\sum_{k=0}^{n} d \times 2^{-k}$, where *d* is a power of 2 and $n \leq 4$ (or whatever maximum number of augmentation dots one chooses). See Sec. 2.3.3.1.) In fact, the algorithm does *not* always generate notatable durations, although problem cases are quite rare; one is given in Fig. 7. When it generates a non-notatable duration, SMUT recursively divides it at the longest integral number of beats of duration shorter than its full duration, unless this would be zero, in which case it divides after the longest undotted duration possible. Some thought shows that this added step always generates notatable durations. In Fig. 7, the basic algorithm generates a duration of five 16ths, which is not notatable. The new step correctly divides that duration into two eighths (which get fused into one quarter) and a 16th.

With the above "post-processing", so far as I know, the rhythm-clarification algorithm works correctly in all simple meter cases for notes. Two problem areas remain, namely compound meters and non-notes, i.e., rests. The first of these is exemplified by Fig. 4*g*. Adding a step to the algorithm between steps 3 and 4 will handle such cases:

(3*a*) In compound meters, divide multiply dotted notes that are at or above the beat level after the first note.

I have not implemented this rule, however.

The second problem area involves treatment of rests. It is clear that the rules for rests are somewhat different from those for notes, both in simple and in compound meters; see Fig. 8 for examples. The differences do not appear hard to formalize, but again I have not done so.[13]

Fig. 9 gives "before" and "after" pictures, showing the results when the above algorithm was applied to a variety of rhythm patterns from Read's book on rhythm notation [READ78]. I think that most musicians would agree that the program behaved correctly everywhere except for measures 2 and 3 of Ex. 2-4A (Read's example number); there the program actually made things worse. These two rhythms probably cannot be handled correctly on a note-by-note basis: instead, the patterns need to be looked at as a whole.[14]

4.4.2. Rhythm Understanding: Beams and Fractional Beams

Another aspect of rhythm understanding that is important in a music-formatting program is deciding what notes to beam, where to put the beams and — least often considered — which way to point fractional beams. SMUT's beaming is controlled at a high level by a variable, BEAM, that affects both the extent of beamable areas and the "aggressiveness" of the beaming. Extent of 0 means each beamable area extends for one beat (as determined from the time signature). Extent of 1 means beamable areas extend for one unit of rhythmic strength one level above the beat, of 2 means for one unit of rhythmic strength two levels above the beat (almost always the whole measure). For example, in 5/4 time, extent of 0 means beamable areas have one quarter-note duration, of 1 means they have alternate half-note and dotted-half durations[15], and of 2 means they cover the whole measure. Regardless of extent, aggressiveness may be set to "aggressive" (beams within an area are broken only when SMUT's simple-minded tests suggest a collision (see Sec. 5.2.1) with an intervening rest, clef, etc., is otherwise very likely), "timid" (beams within an area are broken by any occurence of a symbol that might cause a collision), or "none" (no beams at all).

I have already commented on the standard texts' lack of explicitness about fractional beam direction (see Sec. 2.3.3.2.2). Fig. 10 gives some examples. The basic idea is the same as that of rhythm clarification: fractional beams should be pointed so as to clarify the underlying metric organization of the music. An obvious first guess at a rule is "if a fractional beam is on the first note of a beamed group it points to the right, otherwise it points to the left."[16] Once again the first approximation is not very good: this rule works on Figs. 10*a*, *c*, *f*, and *g*, but not *b*, *d*, or *e*. But with the aid of the rhythmic strength rating system we already have, pointing fractional beams correctly is not difficult.

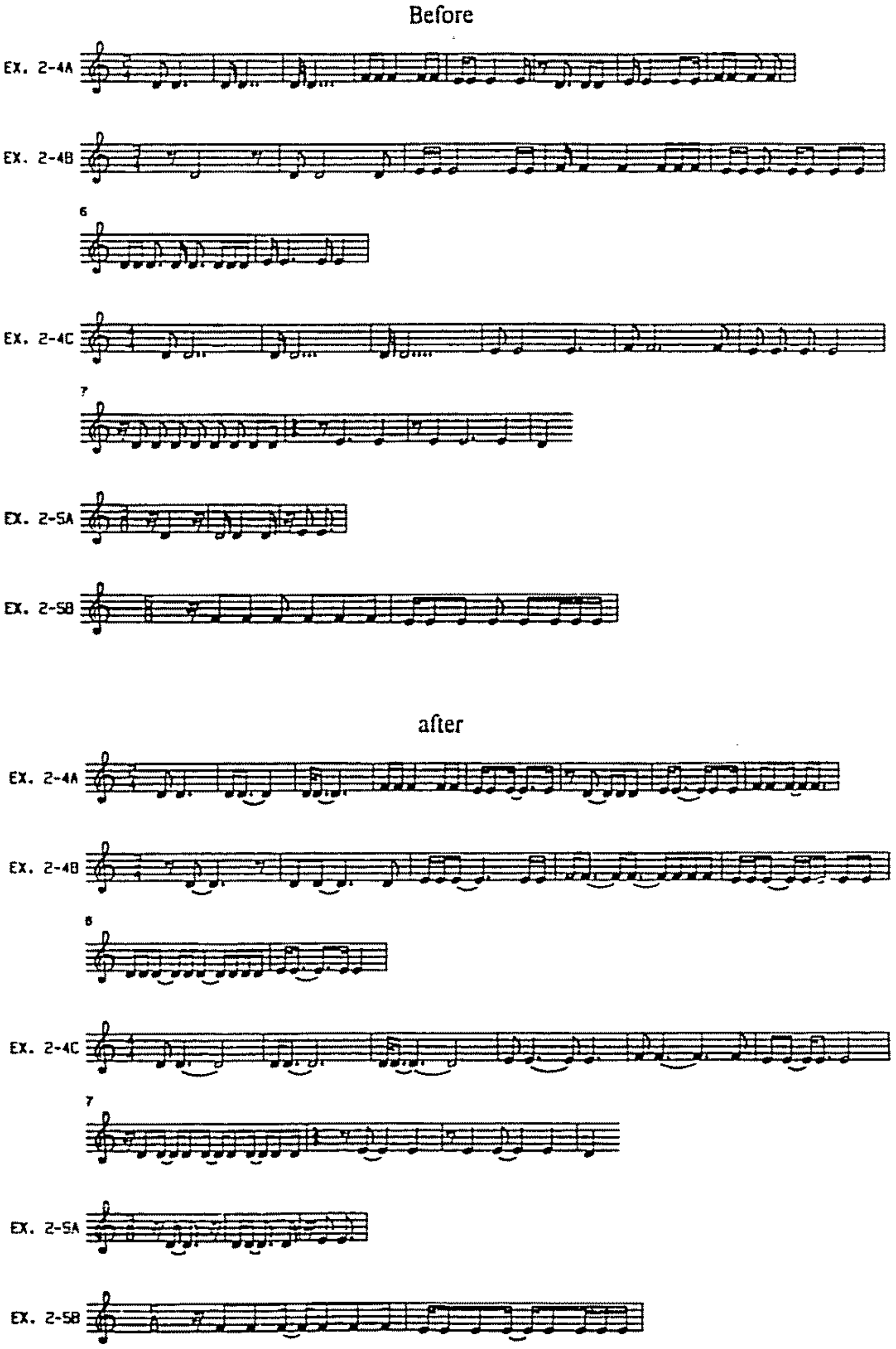


Figure 9. Before and after rhythm clarification

Figure 10. Fractional beam pointing

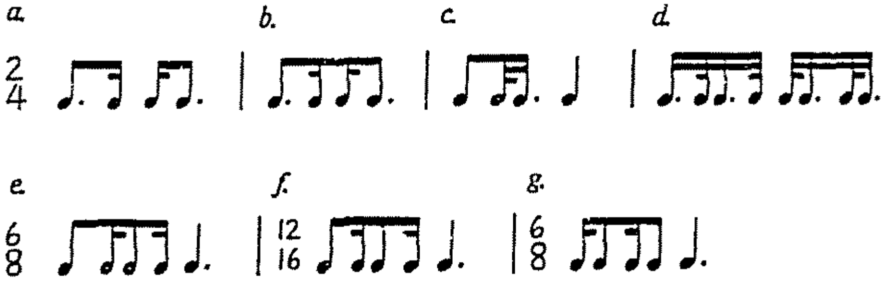
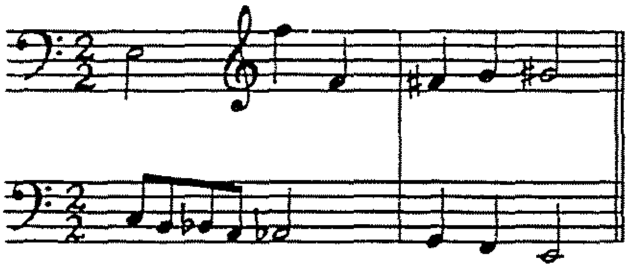


Figure 11. An independent, partially ordered symbol



First, observe that if there is a rest on one side, the beam must point in the other direction, so that we need only consider the case where both adjacent items are notes. Also, the fractional beam cannot point outside of any beamed group it is part of, so if it is on the first or last note of a beamed group it must point to the right or to the left, respectively. (These rules take care of Figs. 10*a* and *c*.) For cases that are still unresolved, let s_1 be the rhythmic strength of the note that has the fractional beam and s_0 and s_2 be the strengths of the preceding and following notes, respectively. Similarly, let t_0, t_1, and t_2 be the attack times of the three notes. Then the algorithm I have implemented is as follows:

(1) Set $s = \min(\max(s_0, s_1), \max(s_1, s_2))$.

(2) Find the first point of strength at least s before t_1; the first such point at or after t_1; and the second such point at or after t_1. These three points divide time into four intervals, each of which we take as closed on the left and open on the right in order to make a partition. We call these intervals, from left to right, 0, 1, 2, and 3.

(3*a*) If t_0 and t_1 are in interval 1 and t_2 is in interval 2, point the beam to the left. If t_0 is in interval 1 and t_1 and t_2 are in interval 2, point the beam to the right. (This correctly handles Fig. 10*b*, *d*, *e*, and *f*.)

(3*b*) If $t_0 = 0$, $t_1 = 1$, and $t_2 = 2$, point the beam to the left. (This handles the only remaining example, Fig. 10*g*.)

(4) If none of the conditions in steps 3*a* and 3*b* are satisfied, increment s by 1 and go to step 2. (There appears to be a danger of an infinite loop here. To detect it, the algorithm actually checks here for $s > 2$ and, if so, quits with an error message; but, to my knowledge, this has never occurred.)

Step 3*b* is rather lacking in intuitiveness. It does, however, handle every case I know of that 3*a* fails on. {Problem: SMUT's failure to consider groupets in assigning rhythmic strength is more likely to cause mistakes here than in rhythm clarification.}

Once it has been decided which notes are to be beamed and which direction fractional beams will point, what remains is to decide whether to position the beams above or below the notes, and then to assign exact y-coordinates to the beginnings and ends of the beams. SMUT does this totally automatically in all cases. Pass I decides whether the beams go above or below the notes, based on the average note vertical position and the most extreme note vertical position. Basically, the algorithm attempts to have the stems on most of the notes and on the most extreme note point in the correct direction.[17] It also assigns tentative y-coordinates to the ends of the beams; final y-coordinates are not assigned until Pass III (see Sec. 4.6).

4.5. THE LAYOUT TASKS AND THEIR ORDERING

This section is based on an identically-titled section of Gomberg's dissertation [GOMB75, pp. 34 – 36]. An unusually thorough treatment of the analogous problems in text setting has been given by Knuth and Plass [KNUT81]; I will refer to that paper repeatedly.

Table 1 contrasts the sequence of basic layout tasks as performed by an engraver or other person formatting music manually, with those performed by SMUT. Two of the terms used here are not well-known except to music engravers. *Casting off* deals with an entire movement and consists of deciding how many measures (not

TABLE 1

Manual			SMUT		
1		Casting-off	1	III	Beams, stems (begin)
2		Vertically position staff	2		Global punctuation
			3		Casting-off
3	W	Punctuation	4	IV	Local punctuation
4		Beams, stems	5		Beams, stems (complete)
5		Ties, slurs	6		Ties, slurs
			7		Vertically position staff

Codes: W: start writing (or engraving); III: beginning of Pass III; IV: beginning of Pass IV.

necessarily an integer: see note 23) will go in each system. This determining of "system breaks" is closely analogous to the determining of line breaks in text [KNUT81]. *Punctuation* is the process of determining the exact horizontal position of every symbol in the score; it involves, among other things, justification of the kind commonly done with text.[18] Referring to text setting, [KNUT81] points out that the term "justification" is now often used for the process of deciding where line breaks in a section of text should go, while formerly it meant the process of adjusting the spacing of a line (after the breaks have been decided) to produce a desired length; this ambiguity causes much confusion.[19]

One problem with the manual process is intuitively obvious: at least some aspects of punctuation should be done before casting off — how can one decide how many measures to put in a system until one knows where the last symbol in each goes? However, without spending a great deal of time making a "practice run", the manual formatter can do punctuation only while writing the finished score, and so she/he is forced to do casting-off first. What happens in practice is that the person does rough punctuation in her or his head while casting off, and is liable to make serious mistakes as a result. This is one of many instances in many domains where computers have an advantage over people because they can do (in [KNUT81]'s term) "late binding". A computer, of course, can do punctuation perfectly "in its head"; SMUT indeed does some of the punctuation — enough to know exactly how much room each measure will need — before casting off. However, it is more convenient to do some of the work later, so SMUT divides punctuation into two parts. (Gomberg does the same thing.) The first step in the whole process is what Gomberg calls *global punctuation*: assigning horizontal positions to symbols as if the entire movement of the score were to be printed on a scroll, i.e., a page of arbitrary width. Casting-off for the actual page width is then performed. Then, in Gomberg's words,

> With the page boundaries decided, a final *local* punctuation must be performed which is a fairly straightforward (nearly) linear expansion or contraction of the exist[ing] punctuation to fit the desired page width.

An even more dramatic difference between manual and automatic techniques has to do with the moment at which the vertical position of the staff on the page is decided. This is again something the manual formatter must do quite early simply so that she/he can start putting ink on paper. A program can (and SMUT does) postpone this till the very end.

4.6. PASS III: GLOBAL PUNCTUATION AND CASTING OFF

On each call, Pass III's subroutine JUSTIF globally punctuates one measure of the score, finalizes beam and slur vertical positions, and optionally determines system breaks, i.e., casts off.

Gomberg draws a distinction between symbols that require horizontal space and symbols that

> are forced to find their space from that which is left over by those that require it . . . Tempo indications, other performance directions, slurs and ties are examples of elements for which space is found after punctuation is completed.

I will term Gomberg's dichotomy that of *independent* and *dependent* x-coordinates. Symbols with dependent x-coordinates are "parasitic" on those with independent coordinates as, for example, the position of a slur is dependent on a pair of notes, one of which determines its left end and one its right end.

This is a good start; however, we need further to subdivide the independent symbols before we can discuss a punctuation algorithm. The need to distinguish a third type of symbol is illustrated by Fig. 11. In our terminology, clefs are certainly independent symbols: they require horizontal space of their own.[20] Nonetheless, the treble clef in the middle of the first measure on the upper staff shows graphic characteristics distinctly different from notes, rests, etc.: this clef does not have any required horizontal position with respect to the second, third, and fourth eighth-notes in the other staff. Instead, it is simply right-justified against the next independent item on its own staff. Let us say, then, that independent symbols may be either *totally ordered* or *partially ordered*. We define the set of totally ordered symbols to include any symbol whose horizontal position is forced with respect to any totally ordered symbol in *any* voice. (Either type of independent symbol has its horizontal position forced with respect to any symbol in its *own* voice.)

The various types of symbols in CMN are categorized as follows:

(1) Independent, totally ordered: notes, rests (other than whole rests), barlines, key signatures, meter signatures.

(2) Independent, partially ordered: clefs, grace notes, pauses.

(3) Dependent: performance directions (depend on one independent symbol), glissandi (depend on two notes or a note and a grace note), groupet accessory

numerals (on two notes or rests), beams (on two notes or two grace notes), slurs and octave signs (on two notes, which may be several measnres apart. The whole rest is a special case: it is normally centered between two barlines.

See also Table 2.

In order to do its punctuation job, JUSTIF requires a list that includes all *attack times*, that is, all times within the measure where a note or rest starts in any voice. In SMUT these times are expressed in multiples of a whole note. So far, this is what [GOMB75] calls the "rhythmic spine". However, JUSTIF also needs entries in its attack time list for all other independent, totally ordered symbols. The problem is to insert these into the list in such a way that (1) the attack times of notes and rests are preserved well enough to allow proper spacing (which is, of course, a function of duration), and yet (2) the symbols we have just inserted — those whose duration is zero — have entries distinct in some way from their neighbors. This could be done in various ways, but SMUT accomplishes it with the following scheme: Let t be, for notes and rests, the actual moment when the item begins in whole notes relative to the beginning of the measure, and for other items, the moment when the preceding note or rest ended. Let d be 1 for key signatures, 2 for time signatures, 3 for notes and rests, 4 for barlines, and 0 otherwise. Then the attack time a is given by

$$a = \text{ceiling}(1000t) + \frac{d}{10}$$

(The numbers 1000 and 10 in this formula are not critical; any similar values would work.) Thus, the attack time of notes and rests is distorted enough to allow inserting other symbols, but not enough to affect spacing noticeably. This scheme places some limitations on what sequences of symbols will give correct results, but the limitations are all outside the syntactic boundaries of CMN: for example, a sequence of symbols involving two clefs without a note or rest in between will not be handled properly, but such a sequence can never occur in CMN. Observe in particular that the sequence "clef, key signature, time signature, note" (as often occurs at the beginning of a piece) will come out in the correct order. Groupets cause no special difficulties.[21] For the first measure of Fig. 11, the attack time list is .2, .3, 125.3, 250.3, 375.3, 500.3, 750.3, 1000.4. For the second measure, it is .3, 167.3, 250.3, 334.3, 500.3, 1000.4.

TABLE 2. Operation Codes and Symbols in Passes III and IV

op code	symbol, subsymbols	JUSTIF type	Representation	Pass
1	staff/clef	IP		
	clef		S,O	
2	key signature	IP		
	accidental		S	
3	time signature	IP	T	
4	note	IT		
	head		O	
	accidental		S	
	ledger line			
	augmentation dot		O	
	note-aligned symbol			
	stem			
	flag			
5	grace note	IP		
	(same components)			
6	rest	IT	S,O	
	rest-related symbol			
	augmentation dot			
7	barline	IT		
	repeat dot		O	
8	beam	D		III
9	performance dir.	D	T	
10	groupet acc.num.	D	T	
11	octave sign	D		
	number		T	
	dotted line			
12	user routine call	-		
13	identification	-	T	IV
14	slur	D	S	III
15	miscellaneous	IP		
16	glissando	D	A	

Codes for Representation: O: outline, S: skeleton, T: text (see Sec. 4.7 for terminology).
Codes for JUSTIF type: IT: independent, totally ordered; IP: independent, partially ordered; D: dependent.
Codes for Pass: III: part of note command on Pass III input file, created by Pass III; IV: written on auxiliary Pass IV input file.

JUSTIF assumes that data describing an entire measure is in an array, MEMB (for "MEMory for Bar"), as a sequentially allocated list of variable-length records. It consists of an initialization section and four main sections. The initialization section does only one task of any interest: if the current measure is the first of any system other than the first, then for each staff it generates a clef and key signature identical to those in effect on that staff at the end of the previous system. The four main sections make four passes through MEMB and do the following:

Section I scans MEMB forward and punctuates type-1 symbols. The algorithm starts with the attack time list. It then:

(1) Assigns "ideal" spacings between consecutive entries in the list according to the formula

$$idealspace = 2.5c\,(t_1 - t_0)^B$$

where t_0, t_1 are consecutive entries in the attack time list, and c and $b = 2^B$ are set with the HCROWD and HBASE control commands, respectively. The defaults are 1.0 for c and 1.520 for b. Here at last is the formula I have alluded to several times. As I said in Sec. 2.3.3.3, what we want is for spacing between consecutive symbols to increase with the time separating them, but more slowly than linearly. (The time separating consecutive symbols is, of course, just the duration of the first.) This formula does the trick as long as $1<b<2$, i.e., as long as $0<B<1$, and a value of b about halfway between comes the closest to giving the spacing of traditionally set music. (Note in particular that if $b=\sqrt{2}$, $B=1/2$, and the formula reduces to

$$idealspace = 2.5c\sqrt{t_1 - t_0}$$

I used this formula in SMUT for several years, but it does not differentiate enough between durations; a slightly larger value of b gives better results.) c, a "crowding" factor, interacts with the rest of the punctuation process to produce an interesting effect. Of course, in this formula, it is simply a scaling factor for the ideal spacing. The interaction results from violations of ideal spacing. In nontrivial music, as we have seen, ideal spacing is constantly violated, but usually only to a slight extent. But if c is made very small, ideal spacing will be so inadequate that it will cease to have any effect on the final setting, and everything will end up simply squeezed as close together as possible, regardless of *b*.[22]

(2) If this does not leave sufficient space between consecutive symbols within any voice, increases it just enough so that there is sufficient space. (JUSTIF knows how much space is needed both backward and forward from the reference point — the *x*-coordinate — of each type of symbol.) {Problem: JUSTIF doesn't *really* know: see Sec. 5.2.4.}

(3) If the resulting measure width overflows the current system and if automatic system breaking was requested, starts a new system. (This is essentially the same as what Knuth and Plass describe for text [KNUT81, p. 1120] as "the standard algorithm for line breaking".)[23] Even if automatic system breaking was not requested, if the measure width overflows the current system badly, JUSTIF gives an error message and starts a new system. If starting a new system results in the old one having zero measures in it, JUSTIF gives an error message. If it does decide to start a new system, JUSTIF pays special attention to slurs, octave signs, and glissandi that cross into the new system. Specifically, it breaks them into two pieces, one for "here" (the end of this system) and one for "there" (the beginning of the new system).[24] It then quits, returning a "start new system" flag. Note that, in this case, MEMB contains partly processed data for the current measure.

Section I also checks the durations of the current measure in all voices for agreement. If the "start new system" flag is set on entry to JUSTIF, it skips Section I and starts with Section II.

Section II scans MEMB backward, punctuates type-2 symbols, and does preprocessing of beams for Section III. Recall that type-2 symbols are right-justified against following symbols; the backward scan is to facilitate this. The beam preprocessing normally consists simply of finding what the slope of each beam is, based on the stems of the end notes. A complicating factor here is that CMN places an ill-defined limitation on beam slopes (according to [GOMB75], 30 degrees). If the calculated slope exceeds the maximum legal slope, the maximum legal slope is used instead. SMUT adopts the usual method of accomplishing the change of slope, which is to adjust the lengths of stems of notes attached to beams; these could not have been calculated before this moment, anyway, since — unless the beam happens to be horizontal — stem end coordinates that will coincide exactly with a beam depend on punctuation.

Section III again scans MEMB backward (with short forward forays) and corrects end-of-stem coordinates for notes (including grace notes) in beamed groups. It uses the slope for each beam decided on in Section II to adjust the y-coordinates of ends of stems to end at the beam, and checks whether each note will then have a long enough stem. If not, it changes the stem end coordinates and moves the beam up or down while leaving its slope unchanged.

Finally, Section IV scans MEMB forward, punctuates type-3 symbols, checks agreement of clefs, key signatures, and time signatures on shared staves, and outputs everything for use by Pass IV. {Problems with the current implementation: JUSTIF needs an image of the page to avoid collisions (see Sec. 5.2.1). JUSTIF should automatically generate "end-of-staff" material (new clef, key signature, and meter, if they are changing) as well as "beginning-of-staff" material. Pass III should do casting-off for the whole movement at a time as Smith's MSS does, not one system at a time. There is a close analogy in text formatting here: most text formatters decide line breaks one line at a time. As Knuth points out [KNUT79, KNUT81], to achieve optimal results, one must consider a whole paragraph at a time, as his TEX does.}

4.7. PASS IV: OUTPUT

Pass IV does all of the actual drawing of the music, one system at a time. It positions each system on the page vertically in accordance with the instructions it receives, starting new pages when necessary; it does "local punctuation" by scaling x-coordinates to right justify the staff (this is disabled in certain circumstances); and it scales y-coordinates for the actual height of the staff. It also handles such details as generating running footers.

The following discussion assumes some knowledge of graphics hardware and systems. For background, see [FOLE82] or [NEWM79].

As I have said, portability was a very important design goal for SMUT. In Pass IV, this means independence of the graphic output device as well as of the CPU. Output-device independence in turn is highly dependent on the availability of standardized subroutine packages for various devices. In the words of Foley and van Dam ([FOLE82], p. 23):

> The main purpose of a device-independent package used in conjunction with a high-level programming language is to induce *application program (and programmer) portability*. This portability is provided in much the same way that a "high-level"

> machine-independent language (such as FORTRAN) provides a large measure of portability: by isolating the programmer from most machine peculiarities and providing language features readily implemented on a wide spectrum of processors.

Foley and van Dam go on to point out the importance of standardization of such packages, and to describe the current state of affairs. Briefly, very substantial effort has now been invested in developing standards for device-independent graphics packages (see [GSPC79] and [ISO81]). However, as of this writing (early 1983), no standard has yet been approved, and what is generally available is still fairly primitive. In any case, when the current work was begun (in 1968) such issues were hardly being considered. As a result, I have written Pass IV in such a way that it can be run on any device with an absolute minimum of support, but can easily be made to take advantage of greater support when available.

Pass IV works in object space (see Sec. 3.3.6) with floating-point coordinates. The Pass IV symbol-drawing routines do all of their drawing by calling four subroutines: RFILL, THLINE, THPLOT, and STRING. The versions I have implemented of these routines have a basic model of the output device that is purely vector-drawing: the only primitives they use are PLOT (move cursor from previous to new position, optionally drawing a vector) and SYMBOL (write a string of text in a specified size and undefined font). As might be expected, nearly all real-world devices support these two very simple primitives, although writing text in any size is often not supported in hardware. Many hardware character generators have size restrictions, so software to write characters in arbitrary size is very widely available.

Now, Pass IV knows the resolution and linewidth of the output device, since they are specified by the user (on the layout line). For many devices, these two parameters have the same value; for pen-and-ink plotters, the linewidth is usually greater than the distance between addressable positions. In any case, with this information, RFILL and THPLOT fill in regions (noteheads, augmentation dots, parts of clefs and rests) by simulating raster-scan hardware, generating their own calls to PLOT to draw lines at intervals of the linewidth. RFILL takes as input a list of vertices bounding the region in question, i.e., an "outline" representation. It can fill in all regions whose boundaries are convex polygons (actually, it can handle some regions with concavities, but SMUT does not rely on this capability). THPLOT draws "thick vectors" — for example, the nonvertical strokes in the sharp and the natural. These thickened lines are in fact convex polygons and could therefore be drawn by

RFILL; I use THPLOT merely for its conceptual similarity to PLOT. THLINE thickens lines in symbols — slurs, bodies of clefs, the rounded part of the flat, etc. — by drawing the center line and edges (and, sometimes, middle lines; see below) from a "skeleton" representation, i.e., specification of a center line and (varying) linewidth. Finally, STRING thickens text simply by writing it repeatedly with small horizontal offsets, namely, successive multiples of the linewidth. Note that the output device's linewidth, not its resolution, is used in all of these cases. Resolution is used directly in only one way, to decide whether to thicken slurs at all or to draw a single "zero-thickness" line. SMUT draws every slur as a circular arc; when it thickens one, it simply uses the same endpoints with slightly different radii of curvature to generate additional arcs, then calls THLINE. {Problem: circular arcs are really not the correct shape. Slurs in engraved music, especially long ones, are more curved at the endpoints than in the middle. Also, slurs can have inflection points, as noted in Sec. 2.3.3.1.1.}

The way in which many symbols are drawn is affected by a "publication quality" switch. If it is set, RFILL takes extra care in filling regions (by drawing the outline as well as making raster scans across the interior) and THLINE takes extra care in thickening everything, especially quarter rests, slurs, and clefs (by drawing middle lines, among other things). This is in an effort to produce the best possible plot quality; however, it is likely to help only on high-resolution vector-drawing devices, especially pen-and-ink plotters, and of course it slows execution down.

This minimal or "least common denominator" output device method is extremely device independent and can produce very good results. For examples of SMUT output produced in this way, compare Figs. 2 and 12: these are, respectively, the score of Bach's so-called "Sloth Canon" printed on a Zeta 3653SX pen plotter, and an instrumental part for the same piece, automatically transposed with a clef change and printed on a Versatec 1200 electrostatic plotter. The Versatec has much lower resolution than the Zeta, particularly visible as jagged edges on slurs and beams. (SMUT took the low resolution into account and did not the thicken the slurs at all, since they would not have looked good.) See also Figs. 13 (printed on the Versatec), 14 (Zeta), 15 (Zeta), 16 (Versatec; a demonstration of some of SMUT's special symbols), and Chapter 3, Fig. 4 (Zeta). However, this device-independent technique has a major disadvantage, namely that it can be very inefficient on more powerful

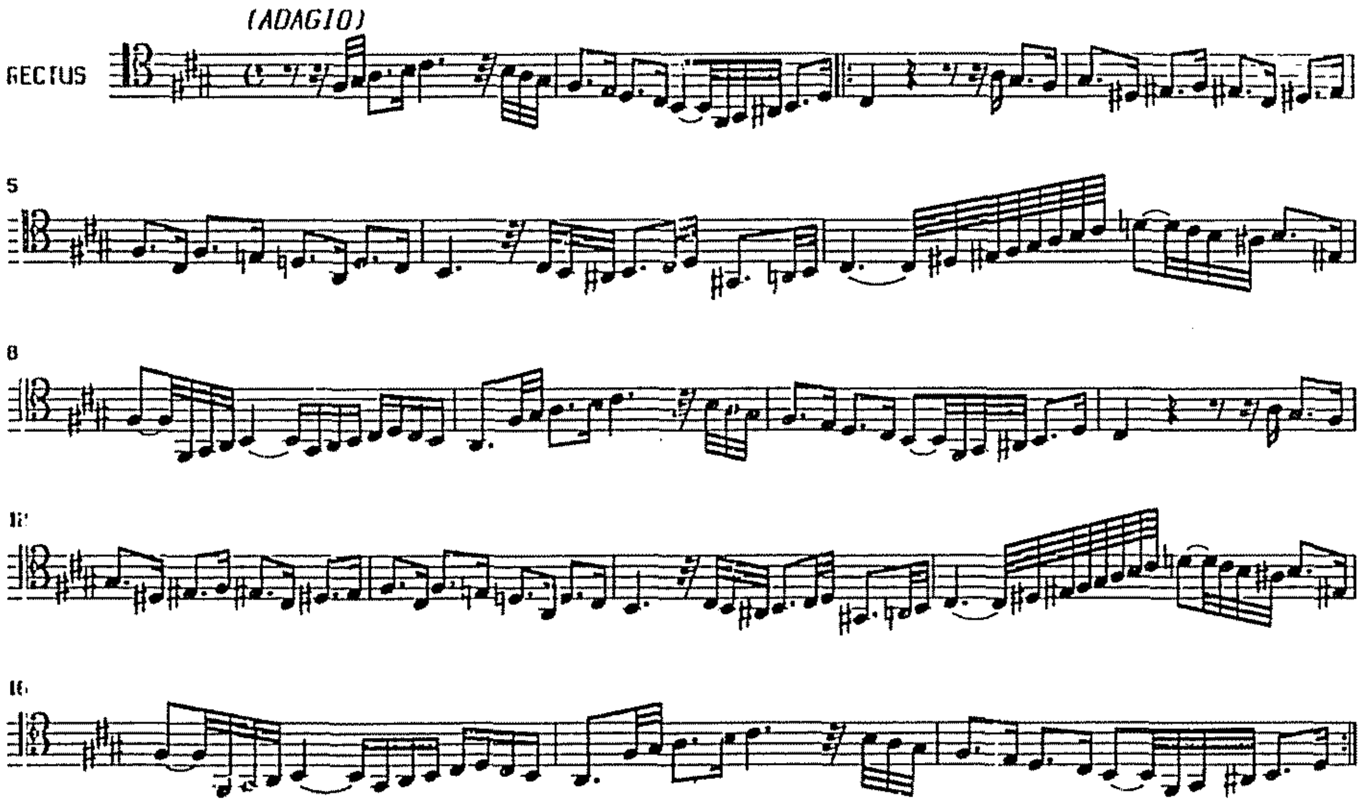


Figure 12. Bach: *Sloth Canon*, a transposed part (set by SMUT).

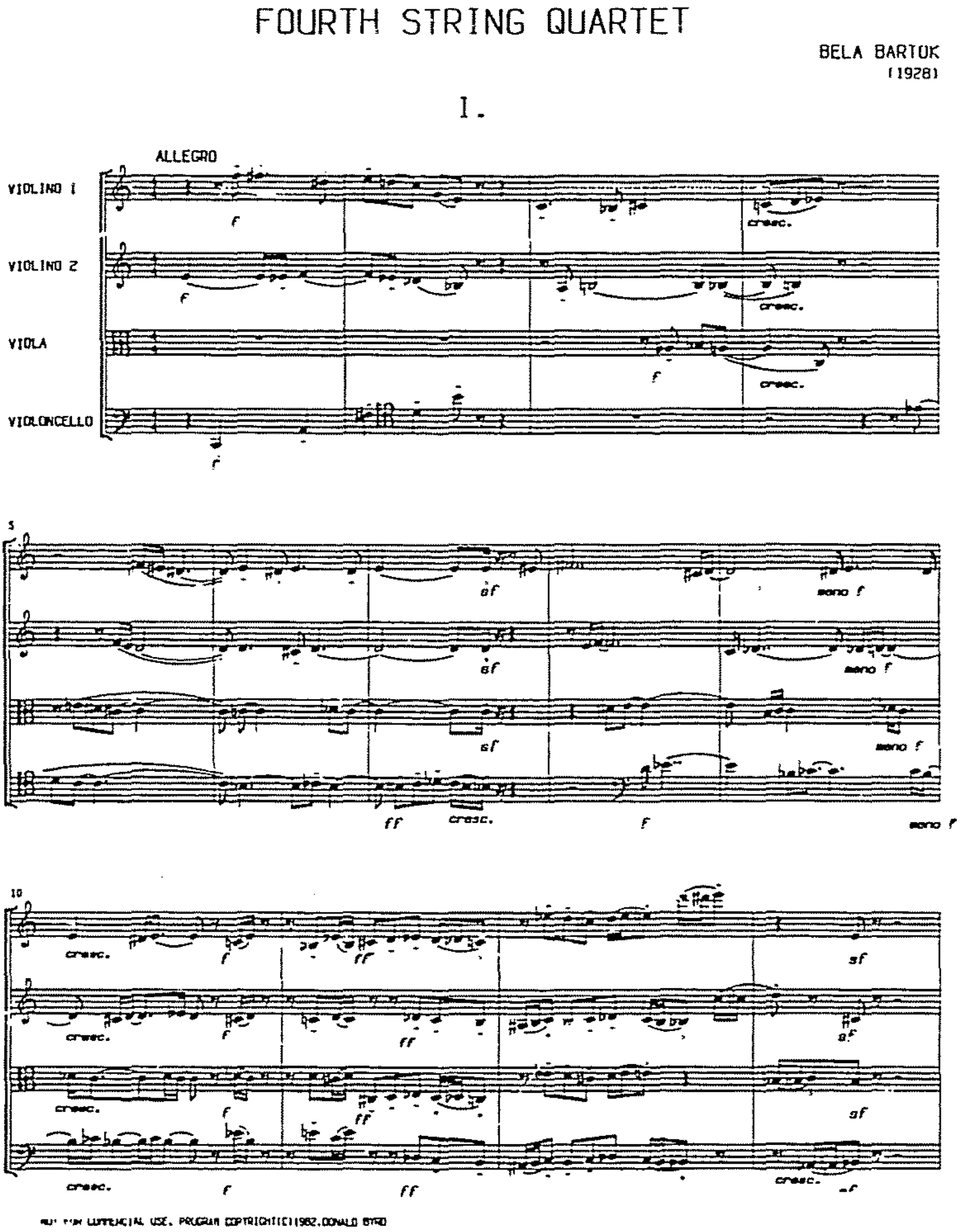


Figure 13. Bartók: *String Quartet No. 4*, I, p. 1 (set by SMUT).

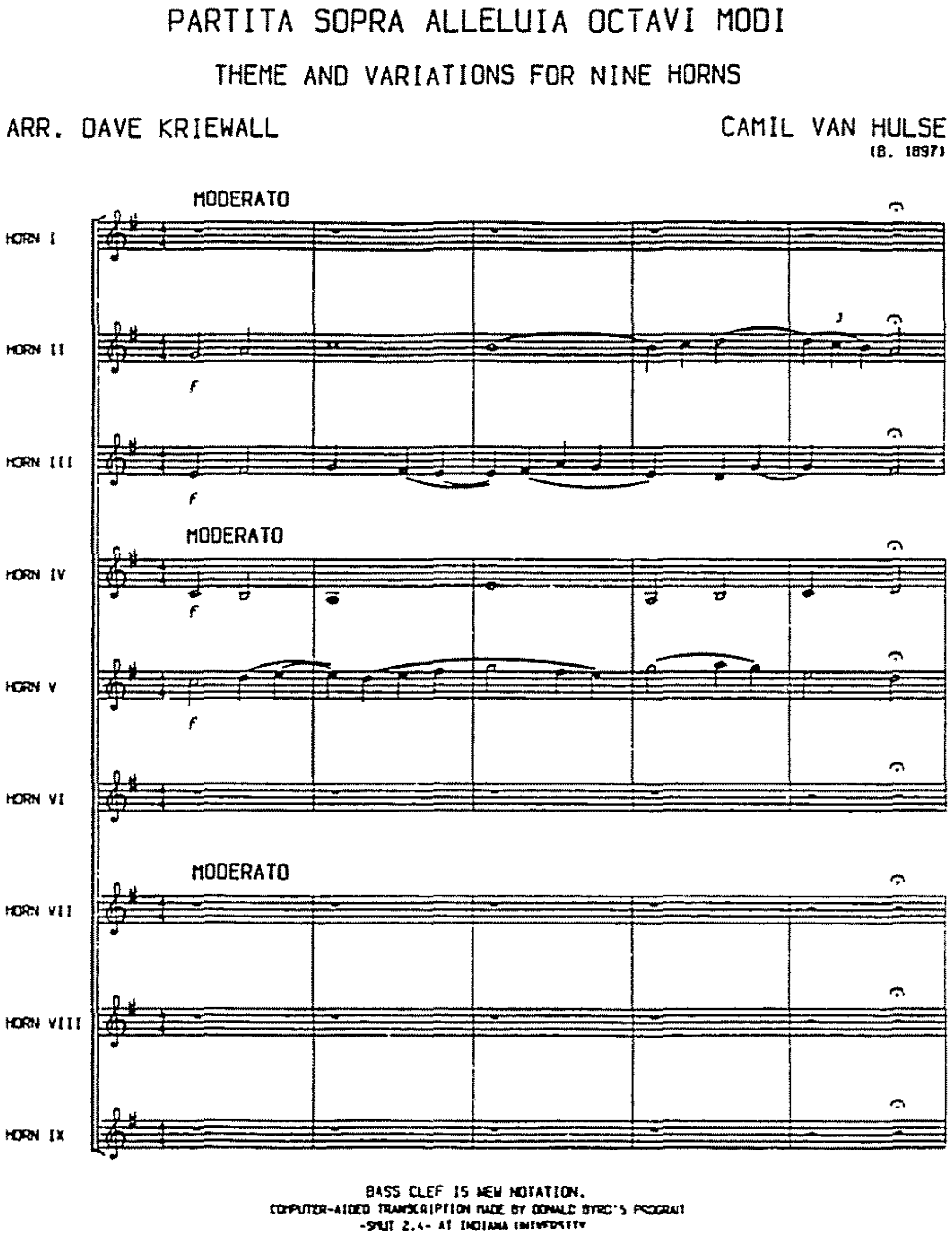

PARTITA SOPRA ALLELUIA OCTAVI MODI
THEME AND VARIATIONS FOR NINE HORNS
ARR. DAVE KRIEWALL
CAMIL VAN HULSE
(B. 1897)
MODERATO
HORN I
HORN II
HORN III
HORN IV
HORN V
HORN VI
HORN VII
HORN VIII
HORN IX
f
BASS CLEF IS NEW NOTATION.
COMPUTER-AIDED TRANSCRIPTION MADE BY DONALD BYRD'S PROGRAM
-SMUT 2.4- AT INDIANA UNIVERSITY

Figure 14. Van Hulse-Kriewall: *Partita Sopra Alleluia Octavi Modi* (set by SMUT).

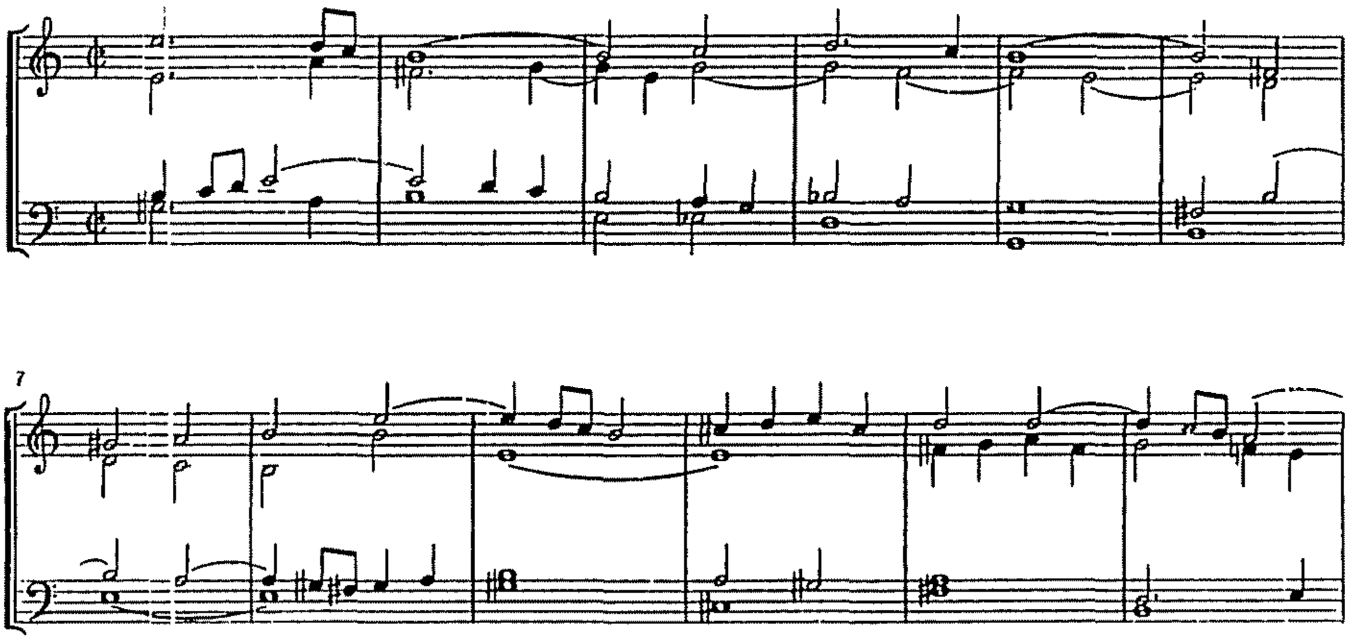


Figure 15. Macque: *Consonanze Stravaganti* (set by SMUT).

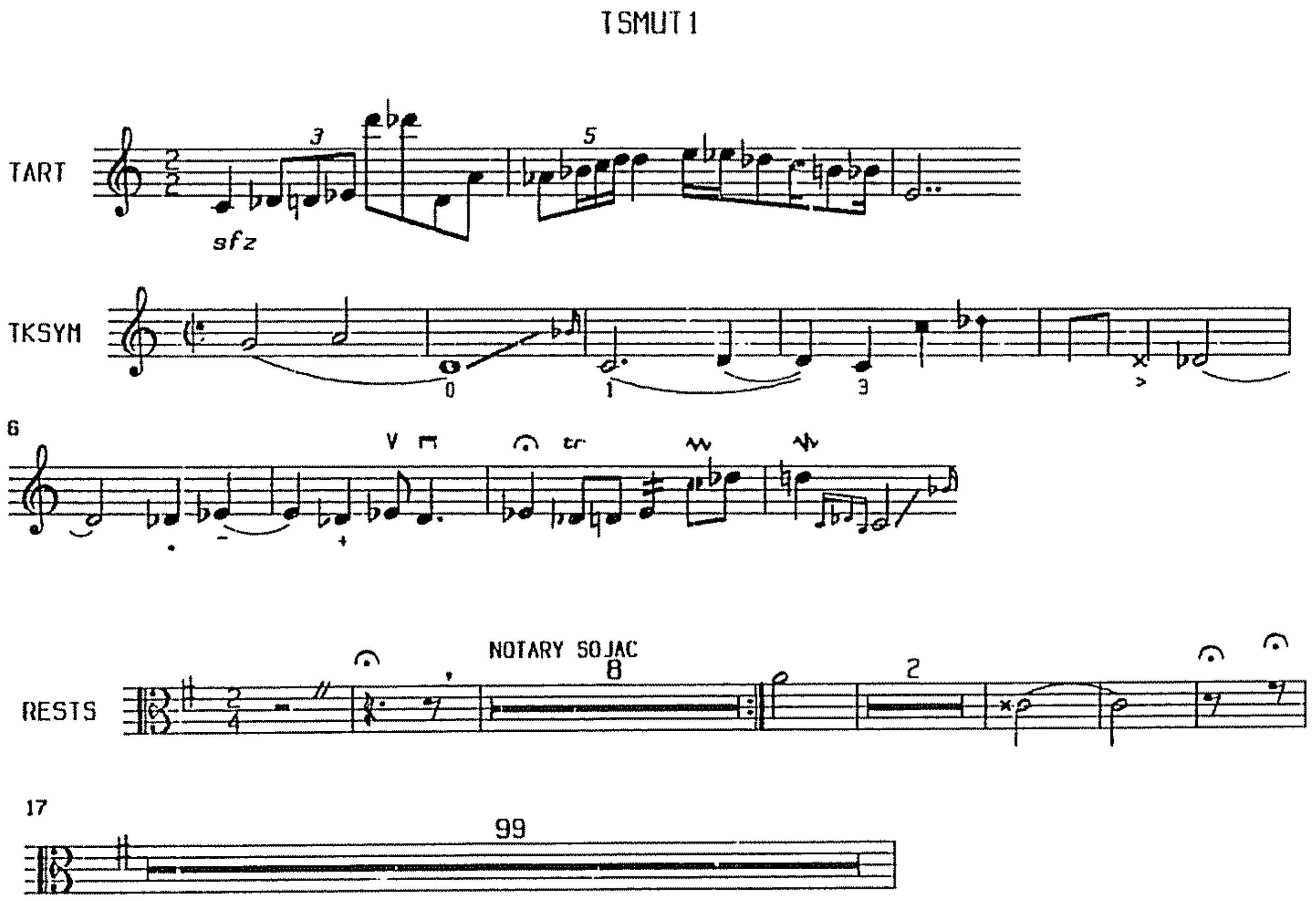


Figure 16. SMUT symbol demonstration

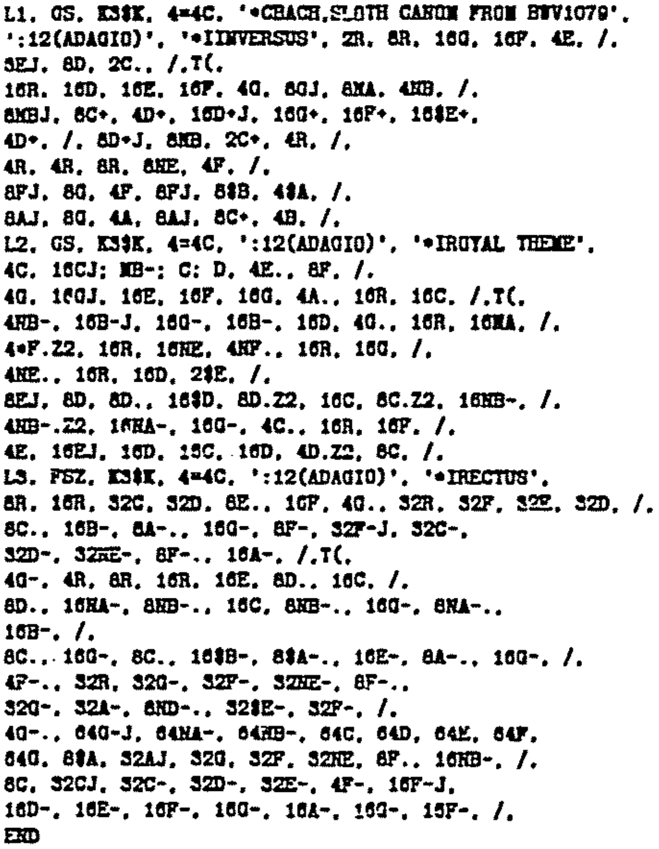
hardware. Such inefficiency can easily be disastrous in interactive graphics, since it is likely to increase response time significantly; but it is rarely serious in a batch program such as SMUT. The particulars described above do involve one compromise with visual quality: thickening of text should really be done not merely with horizontal offsets but also with vertical ones, preferably in a circular pattern. The image of a point under thickening should be a two-dimensional region, not a line. However, the inefficiency of drawing characters $O(n^2)$ times is too great, even for batch use; $O(n)$ repetitions provides generally good quality with much better speed.

As I have suggested, better support than the minimal PLOT and SYMBOL — e.g., hardware area fill or boldface characters, or fast assembly-language routines for area filling (perhaps by generating parallel vectors) — is becoming more and more widely available. When it is available, any or all of RFILL, THPLOT, and STRING can easily be replaced with trivial versions that simply invoke the appropriate operations, so nothing has been lost by the minimal approach, while nearly complete device independence has been gained. THLINE is a little less easy: to take advantage of typical hardware features, it will have to convert from the skeleton representation it receives to a list of vertices in the manner of RFILL. This is still not at all difficult.

4.8. A SAMPLE RUN

In order to make the principles of SMUT's operation clearer, I will now go through a sample run that produces two measures of a three-staff score, specifically the first two measures of the score in Fig. 2. The data for this run appears in MUSTRAN form in Fig. 17. The result of running it through MUSTRAN and SMIRK (see Sec. 3.5) is the SMUT data in Fig. 18. Appendix I gives a complete description of SMUT from the user's viewpoint. However, the format of SMUT commands is quite simple, and I will summarize it here.

A SMUT input file always starts by giving certain global information: it begins with one layout line (having "L" in column 1) and several initialization lines (having various operation codes in column 1; see Table 3). (Also see the description of SMUT's Initialization Phase in Sec. 4.3.) The initialization lines, in this case, run through line 9 and describe only page header and footer information; they are terminated by the line consisting only of "X".



```
L1, GS, K3$K, 4=4C, '*CBACH,SLOTH CANON FROM BWV1079',
':12(ADAGIO)', '*IINVERSUS', 2R, 8R, 16G, 16F, 4E, /.
8EJ, 8D, 2C.. /.T(.
16R, 16D, 16E, 16F, 4G, 8GJ, 8NA, 4NB, /.
8NBJ, 8C+, 4D+, 16D+J, 16G+, 16F+, 16$E+,
4D+, /, 8D+J, 8NB, 2C+, 4R, /.
4R, 4R, 8R, 8NE, 4F, /.
8FJ, 8G, 4F, 8FJ, 8$B, 4$A, /.
8AJ, 8G, 4A, 8AJ, 8C+, 4B, /.
L2, GS, K3$K, 4=4C, ':12(ADAGIO)', '*IROYAL THEME',
4C, 16CJ; NB-; C: D, 4E.. 8F, /.
4G, 16GJ, 16E, 16F, 16G, 4A.. 16R, 16C, /.T(.
4NB-, 16B-J, 16G-, 16B-, 16D, 4G.. 16R, 16NA, /.
4*F.Z2, 16R, 16NE, 4NF.. 16R, 16G, /.
4NE.. 16R, 16D, 2$E, /.
8EJ, 8D, 8D.. 16$D, 8D.Z2, 16C, 8C.Z2, 16NB-, /.
4NB-.Z2, 16NA-, 16G-, 4C.. 16R, 16F, /.
4E, 16EJ, 16D, 16C, 16D, 4D.Z2, 8C, /.
L3, F8Z, K3$K, 4=4C, ':12(ADAGIO)', '*IRECTUS',
8R, 16R, 32C, 32D, 8E.. 16F, 4G.. 32R, 32F, 32E, 32D, /.
8C.. 16B-, 8A-.. 16G-, 8F-, 32F-J, 32C-,
32D-, 32NE-, 8F-.. 16A-, /.T(.
4G-, 4R, 8R, 16R, 16E, 8D.. 16C, /.
8D.. 16NA-, 8NB-.. 16C, 8NB-.. 16G-, 8NA-..
16B-, /.
8C.. 16G-, 8C.. 16$B-, 8$A-.. 16E-, 8A-.. 16G-, /.
4F-.. 32R, 32G-, 32F-, 32NE-, 8F-..
32G-, 32A-, 8ND-.. 32$E-, 32F-, /.
4G-.. 64G-J, 64NA-, 64NB-, 64C, 64D, 64E, 64F,
64G, 8$A, 32AJ, 32G, 32F, 32NE, 8F.. 16NB-, /.
8C, 32CJ, 32C-, 32D-, 32E-, 4F-, 16F-J,
16D-, 16E-, 16F-, 16G-, 16A-, 16G-, 16F-, /.
END
```

Figure 17. MUSTRAN data for the *Sloth Canon*

```
LS   4
H8 C'SLOTH CANON'
H3
H4 C'FROM THE MUSICAL OFFERING, BWV 1079'
H6 R'J.S. BACH'
H4 R'(1685-1750)'
F5 C'SMUT 2.6 SAMPLE PLOT FROM MUSTRAN DATA'
F4 C'DONALD BYRD, WCC, INDIANA UNIV., BLOOMINGTON, IN 47405'
R4 L'SLOTH CANON'
X
*  OJUST
* 85HCROWD
*   NOBAR
*   NODESYNC
*   NOCHROM
S  1  1 -3  4  4 -2  0
*128TLFT128
P1 4  02 8(ADAGIO)
I0 -00 -0 8INVERSUS
R  0  2  0
R  0  8  0
N 33  0 16  0
N 32  0 16  0
N 31  0  4  0TI
B  1
N 31  0  8  0
N 30  0  6  0
N 29  0  2  1
B  1
X
* 85HCROWD
*   NOBAR
*   NODESYNC
*   NOCHROM
S  1  1 -3  4  4 -2  0
*128TLFT128
P1 4  02 8(ADAGIO)
I0 -00 -011ROYAL THEME
N 29  0  4  0TI
N 29  0 16  0
N 28 30 16  0
N 29  0 16  0
N 30  0 16  0
N 31  0  4  1
N 32  0  8  0
B  1
N 33  0  4  0TI
N 33  0 16  0
N 31  0 16  0
N 32  0 16  0
N 33  0 16  0
N 34  0  4  1
R  0 16  0
N 29  0 16  0
B  1
X
```

Figure 18. SMUT input data for the *Sloth Canon*

```
57.  * 85HCROWD
58.  *   NOBAR
59.  *   NODESYNC
60.  *   NOCHROM
61.  S  1  7 -3  4  4 -2  0
62.  *128TLFT128
63.  P1 4  02 8(ADAGIO)
64.  I0 -00 -0 6RECTUS
65.  R  0  8  0
66.  R  0 16  0
67.  N 29  0 32  0
68.  N 30  0 32  0
69.  N 31  0  8  1
70.  N 32  0 16  0
71.  N 33  0  4  1
72.  R  0 32  0
73.  N 32  0 32  0
74.  N 31  0 32  0
75.  N 30  0 32  0
76.  B  1
77.  N 29  0  8  1
78.  N 28  0 16  0
79.  N 27  0  8  1
80.  N 26  0 16  0
81.  N 25  0  8  0TI
82.  N 25  0 32  0
83.  N 22  0 32  0
84.  N 23  0 32  0
85.  N 24 30 32  0
86.  N 25  0  8  1
87.  N 27  0 16  0
88.  B  1
```

TABLE 3. Initialization Operation Codes

op code	meaning
B	Bars-per-system
C	Comment
F	Footer
G	Group voices
H	Header
R	Running footer

The remainder of the input describes each voice in turn. It uses a set of operations completely disjoint from those used in the initialization phase. Column 1 of each line again contains an operation code, but now from the set given in Table 4. The remainder of each line contains parameters for the operation, in a fixed-field format. In this case, the lines in the next group, those beginning with "$", are SMUT control commands. They turn off right justification (11); ask that horizontal spacing be reduced to 85 percent of its usual value (12); disable automatic insertion of barlines and automatic rhythm clarification (13, 14); and specify which of the two possible notations for pitch will be used below (15).

Now the description of the music proper begins. Line 16 asks for a new staff with treble clef, key signature of three flats, and logical time signature of 4/4, but written time signature of "C". (The distinction between "logical" and "real" time signatures is meaningful because, as we saw in Secs. 4.4.1 and 4.4.2, the time signature — in fact, the logical time signature — affects SMUT's automatic rhythm notation procedures.) A command like that on line 17 is always generated by SMIRK in case there is an anacrusis (Sec. 2.3.3.2.3). This one says that the time remaining in the current measure (whose duration is four quarter notes, or 128 128ths) is 128 128th notes: that is, it says that there is no anacrusis. Lines 18 and 19 describe annotations — respectively, a tempo marking and "staff identification". Line 20 at last describes the first time-occupying symbol (a half rest), line 21 the following eighth-rest, and line 22 the first note, the sixteenth-note G4. A tie occurs on line 24

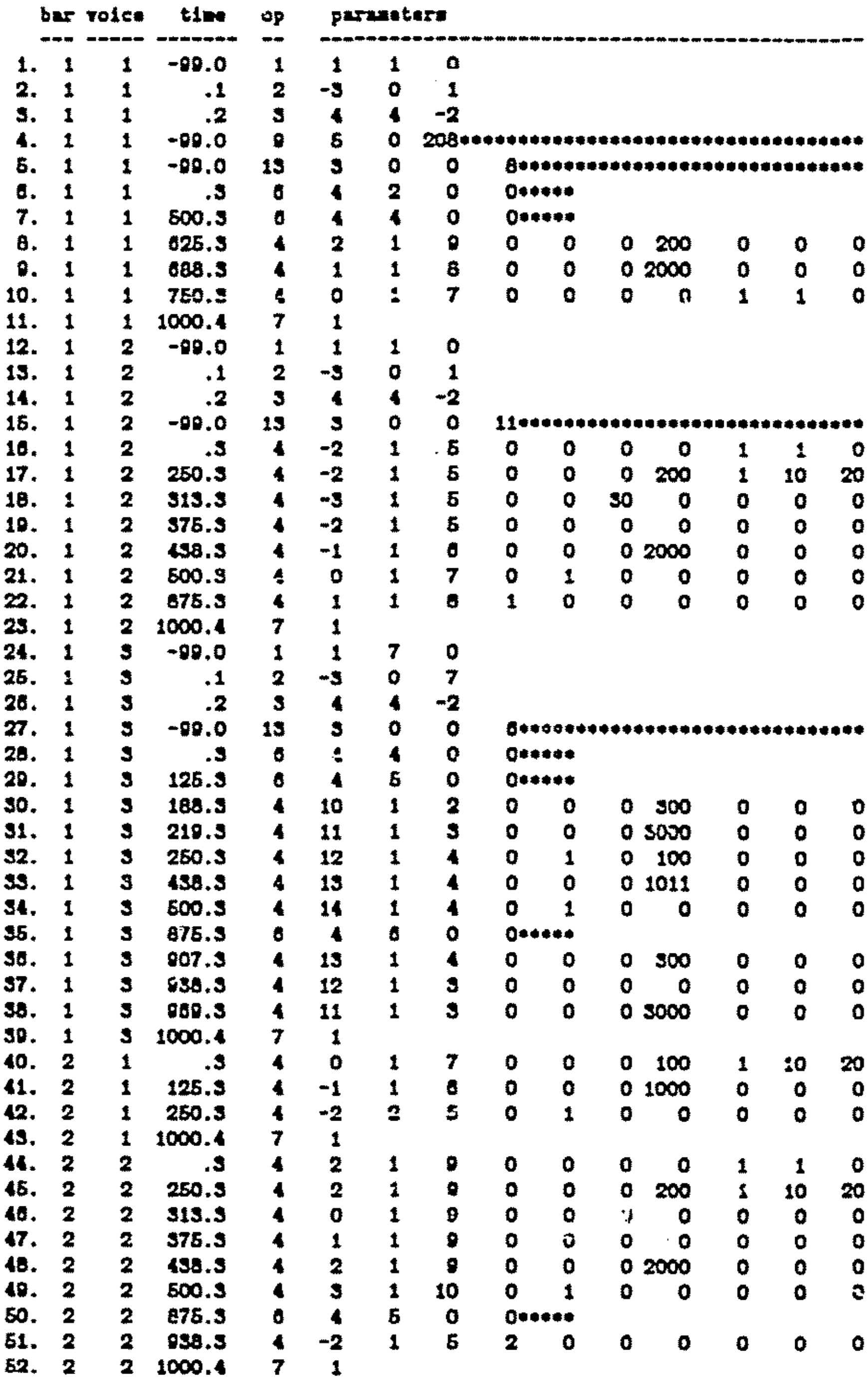
TABLE 4. Pass I Operation Codes

op code	meaning
A	begin Artificial group (groupet)
B	Barline
E	Edit
G	Grace note
I	Identification/indent
M	Miscellaneous symbol
N	Note
P	Performance direction
R	Rest
S	Staff-clef-key signature-meter
T	ocTave sign
U	call User routine
*	control

("TI"), and the data for voice 1 are terminated on line 30 by the "X". The only interesting new feature in the other voices is the accidental on line 41 (a natural, indicated by the "30").

This is, then, a straightforward description of the music that says nothing explicit about symbol positions, not even stem directions. It does not say whether the ties should be above or below the notes, let alone give exact positions for them. Beams are specified to an even lesser degree — in fact, there is absolutely no information about them! The data (other than global information at the beginning) are organized linearly by voices: all of voice 1, then all of voice 2, then all of voice 3.

The Pass III input file illustrated, slightly truncated, in Fig. 19 shows the effects of two major transformations, namely those of Passes I and II. As I said above, Pass II simply reorders the records, sorting them first by measure and then by instrument. Within each instrument, they remain in their original order, but with some



```
   bar voice   time   op    parameters
   --- ----- -------  --   ------------------------------------------------
 1.  1    1   -99.0    1    1    1    0
 2.  1    1      .1    2   -3    0    1
 3.  1    1      .2    3    4    4   -2
 4.  1    1   -99.0    9    5    0  208***********************************
 5.  1    1   -99.0   13    3    0    0    8*****************************
 6.  1    1      .3    6    4    2    0    0*****
 7.  1    1   500.3    6    4    4    0    0*****
 8.  1    1   625.3    4    2    1    9    0    0    0  200    0    0    0
 9.  1    1   688.3    4    1    1    8    0    0    0 2000    0    0    0
10.  1    1   750.3    4    0    1    7    0    0    0    0    1    1    0
11.  1    1  1000.4    7    1
12.  1    2   -99.0    1    1    1    0
13.  1    2      .1    2   -3    0    1
14.  1    2      .2    3    4    4   -2
15.  1    2   -99.0   13    3    0    0   11*****************************
16.  1    2      .3    4   -2    1    5    0    0    0    0    1    1    0
17.  1    2   250.3    4   -2    1    5    0    0    0  200    1   10   20
18.  1    2   313.3    4   -3    1    5    0    0   30    0    0    0    0
19.  1    2   375.3    4   -2    1    5    0    0    0    0    0    0    0
20.  1    2   438.3    4   -1    1    6    0    0    0 2000    0    0    0
21.  1    2   500.3    4    0    1    7    0    1    0    0    0    0    0
22.  1    2   875.3    4    1    1    8    1    0    0    0    0    0    0
23.  1    2  1000.4    7    1
24.  1    3   -99.0    1    1    7    0
25.  1    3      .1    2   -3    0    7
26.  1    3      .2    3    4    4   -2
27.  1    3   -99.0   13    3    0    0    8*****************************
28.  1    3      .3    6    4    4    0    0*****
29.  1    3   125.3    6    4    5    0    0*****
30.  1    3   188.3    4   10    1    2    0    0    0  300    0    0    0
31.  1    3   219.3    4   11    1    3    0    0    0 5000    0    0    0
32.  1    3   250.3    4   12    1    4    0    1    0  100    0    0    0
33.  1    3   438.3    4   13    1    4    0    0    0 1011    0    0    0
34.  1    3   500.3    4   14    1    4    0    1    0    0    0    0    0
35.  1    3   875.3    6    4    6    0    0*****
36.  1    3   907.3    4   13    1    4    0    0    0  300    0    0    0
37.  1    3   938.3    4   12    1    3    0    0    0    0    0    0    0
38.  1    3   969.3    4   11    1    3    0    0    0 3000    0    0    0
39.  1    3  1000.4    7    1
40.  2    1      .3    4    0    1    7    0    0    0  100    1   10   20
41.  2    1   125.3    4   -1    1    6    0    0    0 1000    0    0    0
42.  2    1   250.3    4   -2    2    5    0    1    0    0    0    0    0
43.  2    1  1000.4    7    1
44.  2    2      .3    4    2    1    9    0    0    0    0    1    1    0
45.  2    2   250.3    4    2    1    9    0    0    0  200    1   10   20
46.  2    2   313.3    4    0    1    9    0    0    0    0    0    0    0
47.  2    2   375.3    4    1    1    9    0    0    0    0    0    0    0
48.  2    2   438.3    4    2    1    9    0    0    0 2000    0    0    0
49.  2    2   500.3    4    3    1   10    0    1    0    0    0    0    0
50.  2    2   875.3    6    4    5    0    0*****
51.  2    2   938.3    4   -2    1    5    2    0    0    0    0    0    0
52.  2    2  1000.4    7    1
```

Figure 19. SMUT Pass III input for the *Sloth Canon*

significant changes, as we shall see. (Henceforth, we shall adopt the convention that an expression like "line III-24" refers to line 24 of Pass III input. Pass I input appears in Fig. 18, Pass III input in part in Fig. 19, and Pass IV input in part in Fig. 20.) Not surprisingly, the most interesting case is that of the note. Line I-22 describes the first note in the top voice; it is replaced by line III-8, then IV-7 (op code 4 is a note in both passes III and IV; see Table 2). Here is a summary of the information in the three forms. The coordinate systems were described in Sec. 4.3.

I-22 pitch=33 0 (E4, no accidental), duration=16 0 (16th-note with 0 dots).

III-8 time=625.3, y=2, head type=1, y-stem=9, 0 flags, 0 dots, accidental type=0, beam activity=200 (begin 2 beams), begin 0 slurs and ties, slur direction=0, slur modifiers=0.

IV-7 x=8.670, y=250, head type=1, y-stem=1120, 0 flags, 0 dots, accidental type and position=0.

Pass IV is a straightforward interpreter that calls external subroutines to draw all of the music; its principles were described in Sec. 4.7.

NOTES

[1] A slight area of fuzziness is stem direction for notes on the middle of the staff. General opinion here is that this was formerly a matter of choice but that down-stems are now *de rigueur*.

[2] As mentioned in Sec. 2.3.3.1.1, ties are logically somewhat different from slurs, but, graphically, the possible shapes and positions of ties are a subset of the possible shapes and positions of slurs. In fact, after Pass I, SMUT does not distinguish between the two. Unless otherwise specified, the term "slur" henceforth should be taken to mean "slur or tie" (however, "slur or tie" should not be taken to mean "slur or tie or tie").

[3] Actually, the argument that this particular group *had* to be beamed down is rather weak, but the same tie placement situation would have arisen if the second note in the beamed group were much higher, in which case the beaming would unquestionably have had to be down.

[4] This rule is particularly reasonable for two-note slurs (and all ties), since it avoids any intersection between the slur/tie and the stem of the second note.

[5] The problems of aligning multiple voices in parts are far less serious, when they occur at all, than in scores. On the other hand, placing page turns (see Sec. 2.3.6.4) is very hard in both, but

```
      bar  voice     x     op  parameters
      ---  -----  ------   --  ---------------------------------------
  1.   1     1      .100    1     1      1       0*********************
  2.   1     1     1.350    2    -3      0       1*********************
  3.   1     1     2.860    3     4      4      -2*********************
  4.   1     1     3.070    9     5      0     208*********************
  5.   1     1     4.060    6   500      2       0    1000**************
  6.   1     1     6.065    6   500      4       0    1000**************
  7.   1     1     8.670    4   250      1    1120       0      0      0
  8.   1     1     0.000    8  8670    920    9070     800      2      0
  9.   1     1     0.000    8  8670   1120    9070    1000      2      0
 10.   1     1     9.070    4   125      1    1000       0      0      0
 11.   1     1     9.470    4     0      1     875       0      0      0
 12.   1     1    11.875    7     1      1     875       0      0      0
 13.   1     2      .100    1     1      1       0       0      0      0
 14.   1     2     1.350    2    -3      0       1       0      0      0
 15.   1     2     2.860    3     4      4      -2       0      0      0
 16.   1     2     4.060    4  -250      1     625       0      0      0
 17.   1     2     0.000   14  4110   6015    -500    -500    -20      0
 18.   1     2     6.065    4  -250      1     625       0      0      0
 19.   1     2     6.665    4  -375      1     687       0      0     30
 20.   1     2     7.265    4  -250      1     719       0      0      0
 21.   1     2     0.000    8  6065    425    7665     550      2      0
 22.   1     2     0.000    8  6065    625    7665     750      2      0
 23.   1     2     7.665    4  -125      1     750       0      0      0
 24.   1     2     8.065    4     0      1     875       0      1      0
 25.   1     2    10.075    4   125      1    1000       1      0      0
 26.   1     2    11.875    7     1      1    1000       1      0      0
 27.   1     3      .100    1     1      7       0       1      0      0
 28.   1     3     1.350    2    -3      0       7       1      0      0
 29.   1     3     2.860    3     4      4      -2       1      0      0
 30.   1     3     4.060    6   500      4       0    1000*******      0
 31.   1     3     4.665    6   500      5       0    1000*******      0
 32.   1     3     5.255    4  1250      1     170       0      0      0
 33.   1     3     0.000    8  5265    570    5665     690     -2      0
 34.   1     3     0.000    8  5265    370    5665     490     -2      0
 35.   1     3     0.000    8  5265    170    5665     290     -2      0
 36.   1     3     5.665    4  1375      1     220       0      0      0
 37.   1     3     6.065    4  1500      1     500       0      1      0
 38.   1     3     0.000    8  7665    700    7634     700     -2      0
 39.   1     3     0.000    8  6065    500    7665     500     -2      0
 40.   1     3     7.665    4  1625      1     500       0      0      0
 41.   1     3     8.065    4  1750      1     500       0      1      0
 42.   1     3    10.075    6   500      6       0    1000*******      0
 43.   1     3    10.775    4  1625      1     420       0      0      0
 44.   1     3    11.175    4  1500      1     358       0      0      0
 45.   1     3     0.000    8 10775    820   11575     695     -2      0
 46.   1     3     0.000    8 10775    620   11575     495     -2      0
 47.   1     3     0.000    8 10775    420   11575     295     -2      0
 48.   1     3    11.575    4  1375      1     295       0      0      0
 49.   1     3    11.875    7     1      1     295       0      0      0
 50.   2     1     0.000   14  9520  12445    -250    -250    -20      0
 51.   2     1    12.495    4     0      1     875       0      0      0
 52.   2     1     0.000    8 12495    875   13100     750      2      0
```

Figure 20. SMUT Pass IV input for the *Sloth Canon*

follows different principles. The difficulty of placing page turns is, for the time being, a moot point: no computer formatting system I know of, including SMUT, even attempts to place them automatically.

[6] Prototype programs for both purposes have been written at Indiana University: the editor by Rosalee Nerheim, the program to combine music and text by me.

[7] However, SMUT does not support all of the levels I mention in Sec. 2.3.6.2.3.

[8] The reverse capability — fusing several consecutive rests or tied notes into one — would be both useful and relatively easy to implement, but SMUT does not provide it.

[9] This is especially surprising in two books that concentrate on 20th-century notation, [STON80] and [READ78]. Stone clearly recognizes the great importance of rhythm notation in 20th-century music; Read, indeed, is writing exclusively on rhythm notation. Nonetheless, neither includes anything like a clear statement of principles or a comprehensive set of examples on subdividing notes to clarify the rhythm. (For more on Read, see note 14 below.)

[10] For notation purposes, there is no need to establish a hierarchy of *measures*, since in CMN every measure must begin with a note or rest and, therefore, no decision need ever be made about whether to divide notes or rests at the barline. (Sec. 2.5 and particularly note 51 of Chapter 2 cite instances of dotted notes with the notehead before the barline and the dot after, a practice that was not uncommon in the 16th and 17th centuries but is now very rare. Also, measures with no note or rest on the downbeat can result from groupets that cross the barline (as in Chapter 2, Fig. 36, oboe), but such groupets are arguably not CMN.) However, a hierarchy of measures can be important for other purposes; see note 11 below.

[11] Allen Winold has pointed out that this scheme can easily be extended to rank strengths of downbeats of measures, something of value to music theorists for nonnotational purposes. One can simply use consecutive positive integers starting with, say, 5 for successively stronger downbeats.

[12] This phenomenon is not at all rare; in fact, it may be recalled that we used the need to notate such times as the basic justification for the existence of groupets (Sec. 2.3.3.1).

[13] Read, Ross, and Stone all give some sort of usage rules for rests; however, none of them compares them to the rules for notes or makes it at all easy for a reader to do so. Furthermore, all make the identical mistake of giving their rules in too concrete a form, referring to specific rest durations, when what they clearly mean is rest durations relative to the beat and, therefore, to the time signature. See [READ69], p. 100; [ROSS70], p. 179; [STON80], p. 133.

[14] Read also gives two versions of each example, similar to my "before" and "after" versions, which he calls "traditional format" and "modern format", and says that the "modern" versions have "equal clarity and pragmatism". I disagree, and so do most musicians I have asked.

[15] SMUT assumes that meters with five beats are divided 2+3, although 3+2 is also quite

possible.

[16] In essence, this is the rule given in [DONA63].

[17] [GOMB75] gives a more elaborate and probably better algorithm (pp. 72 – 74).

[18] The similarity between punctuation in music setting and line-breaking and justification problems in setting "polyglot Bibles" is especially striking. Knuth and Plass remark [KNUT81, p. 1166]:

> One of the most difficult challenges faced by printers over the years has been the typesetting of "polyglot Bibles" — editions of the Bible in which the original languages are set side by side with various translations — since special care is needed to keep the versions of various languages synchronized with each other.

Knuth and Plass discuss several such Bibles printed between 1517 and 1657. Needless to say, these were all set by hand, and Knuth and Plass say almost nothing about what automated setting of such texts would require.

[19] The terms *filling* and *adjusting*, respectively, are sometimes used for these functions, e.g. in the well-known UNIX text formatters NROFF and TROFF and their documentation [OSSA76, KERN78b].

[20] Instances are occasionally found in CMN in which a clef shares horizontal space with a note (see Sec. 2.5 for examples). It is debatable whether this is really correct; it is certainly not standard.

[21] With respect to adding groupets to the spine, Gomberg incorrectly states [GOMB75, p. 46]: "One measure of the difficulty involved is that no existing automated process appears to even try to print groupets." Actually, early versions of SMUT were printing groupets well before 1975 (see [BYRD74]) — although not the *nested* groupets that Gomberg tries to handle.

[22] [GOMB75] describes in detail a somewhat different and considerably more complex spacing method that is apparently the standard method used by music engravers. The computer notation systems of Armando Dal Molin [DALM78] and Leland Smith [SMIT73] use a "pseudo-Fibonacci series" approach: when the duration is doubled, the ideal spacing goes to the next Fibonacci number (times a constant). For reasons I do not fully understand, all three methods actually produce very similar results, and I doubt if anyone but an engraver would care about the differences. My method has the advantage over the others of having two parameters that might usefully be varied, for example to set music with very tight spacing.

[23] There are two differences. First, SMUT does not "hyphenate": the equivalent in CMN, breaking measures across systems, is indeed possible, but is far more undesirable and rarely done. A brief discussion of such phenomena appears in Scott Kim's book [KIM81, p. 93]. Second, except under very special circumstances, SMUT never compresses: as far as it is concerned, the ideal spacing is the minimum — which is not really a good assumption.

[24] The equivalent in formatting most natural languages — breaking lines — can at worst involve breaking a word and adding a hyphen to the first part. This requires a dictionary or rules to decide where to hyphenate, but is otherwise much simpler. In German, breaking a word across lines may also result in changes to its spelling.

5
Conclusions

5.1. INTRODUCTION

In this concluding chapter, I will try to shed some light on the future of computer music setting by criticizing both my work and that of others. What approaches don't work well and what approaches do? Why? I will discuss what seem to me the most important problem areas in detail, but will touch on other areas only briefly. The chapter will also be fairly speculative.

5.2. SYMBOL PLACEMENT

Ideally, symbol positioning in CMN should be done so that the notation not only conveys the correct meaning, but does so with as little effort on the reader's part as possible. But it should be obvious by now that following the explicit rules of CMN, as given in the standard texts, is not even sufficient to guarantee correctness, much less maximum ease of reading. This is strongly suggested by Sec. 2.5, "Counterexamples and 'Fully Automatic High Quality Music Notation' ". It was also the burden of my statement in the Introduction (Sec. 1.3) that one of the goals of my work was

> to gather explicit knowledge about one of the most sophisticated notational systems in existence, namely CMN. There are of course many books on music notation, but all, or nearly all, were written by musicians for musicians, and the natural result is that they make huge assumptions . . . The closest thing I know of to an explicit statement . . . is still no more than, say, 5% explicit.

A recent paper surveying document-formatting systems of many types includes a section entitled "Relations among Concrete Objects" [FURU82, pp. 458 – 59] that briefly discusses some of the formatting issues that arise in symbol placement for CMN. It brings up and criticizes some of the ideas for "constraints" found in such systems as TEX (glue between adjacent objects), IDEAL (equations involving points on arbitrary objects), and ThingLab ("general" constraints). According to [FURU82], the technique used in ThingLab "includes equation solving as a special case, but is much more general [than IDEAL's technique] and may be used with constraints that are not numeric." Two references for constraint systems that [FURU82] does not mention are [KNUT79], which discusses Knuth's well-known equation-based system METAFONT, and [GIPS75], which discusses still another type of constraint system, the shape grammar with coordinates. Some of these are promising for positioning symbols in CMN, especially in conjunction with other approaches, but it is very doubtful that any of them "as is" can solve the problem.

5.2.1. An Unsolved Problem: Avoiding Perceptual Collisions

One important requirement of music setting that is still far from explicit is that of *avoiding collisions*. A collision might be defined as a situation where two symbols overlap or touch. This requirement is especially interesting because it has close relatives in other areas of computer graphics, some of which we will only touch on and some of which we will consider in detail: in realistic three-dimensional graphics (the "hidden-surface" problem), in cartography (the problem of labelling map features), in design (the problem of positioning physical objects, e.g., components and paths in an electronic circuit), and in an area that is not ordinarily considered part of graphics at all, namely, typesetting text (the "kerning" problem). The latter case is different from, and much simpler than, all the others in that it is essentially one-dimensional, while they are two-dimensional; nonetheless, it is worth thinking about.

As we saw in Secs. 2.4 and 2.5, many types of collision are perfectly acceptable in CMN, especially where both symbols are (in some extended sense) lines. In the latter case the English language has a more specific term than "collision", namely *intersection*. For example, intersections of slurs and ties with stems, accidentals, barlines, etc., are quite common, especially in crowded areas (see Sec. 2.5). These collisions are acceptable because it is easy to see the separate symbols involved. On the other hand, some situations where symbols do not actually touch but merely come close to each other are objectionable. So the term "collision" is simultaneously too inclusive (since some intersections are acceptable CMN) and not inclusive enough (since some near-misses are not acceptable). We need a better term. Let us, then, rename a situation where two symbols overlap or touch a *literal collision*, and call a situation where two symbols overlap, touch, or come close to touching in such a way that they cannot easily be separated by a trained eye a *perceptual collision*. (Thus neither type of collision is a subset of the other.) Intersections are always literal collisions, but are not necessarily perceptual collisions. In CMN, of course, we are concerned exclusively with avoiding perceptual collisions. Some clearcut examples of literal collisions that are very likely to be perceptual collisions would be: slurs cutting through noteheads or groupet auxiliary numerals, and performance directions being superimposed on noteheads or beams. (In fact, these are all very rare in published music.) Perceptual collisions that are not literal collisions are harder to describe verbally; Fig. 1 shows three, arranged in order of increasing complexity. Fig. 1*a* is confusing at first glance because the accidental, which can only belong to the following note, is positioned closer to the preceding note. (A similar case is in Chapter 4, Fig. 13, m. 7, violins, where a staccato dot is too far from its note and too close to a nearby dynamic marking.) In Fig. 1*b*, an augmentation dot on one note appears to be a staccato dot on another. (A similar case in published music is in Bach's *Nun komm', der Heiden Heiland*, Peters edition, m. 8.) Finally, Fig. 1*c* is an extreme case, suggested by Hofstadter [HOFS83a], in which "correct" placement of staccato dots results in one being positioned *inside* an adjacent notehead! How can it be insured that such things do not occur?

Actually, we can distinguish two classes of domain, corresponding to the two types of collisions. In the first, the "pictures" are intended for human viewing: this involves perceptual collisions. In the second, they are intended for use in a machine: this requires a slight generalization of the notion of "literal collisions". For example, in electronic design, layouts of printed-circuit boards and integrated circuits fall into the second category. They are intended to be used in making machines, and so they must satisfy absolutely rigid criteria such as minimum line widths and spacings: see, e.g., [MEAD80]. But guaranteeing minimum spacing between symbols can be reduced to avoiding literal collisions simply by surrounding each symbol with an "envelope" and then looking for literal collisions between the envelopes. I will call violations of this type of spacing constraint *literal near-collisions*. Literal collisions are then a special case of literal near-collisions in which the envelopes are identical to their objects.

Circuit schematics and most types of maps are much more like CMN in that they are intended for human consumption, so that the only criterion is

Figure 1. Perceptual collisions that are not literal collisions

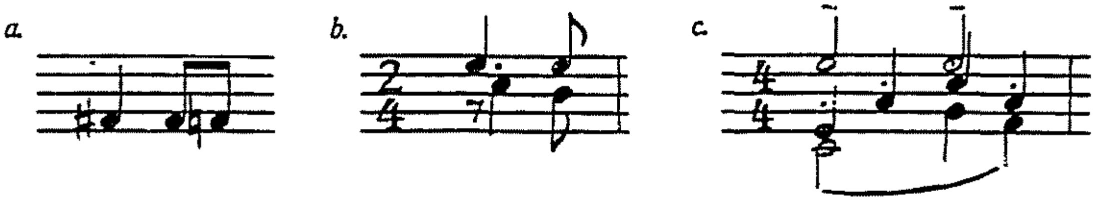


Figure 2. Collision avoidance in text setting

(a)

SAVANT Wonder Whiz

(b)

SAVANT Wonder Whiz

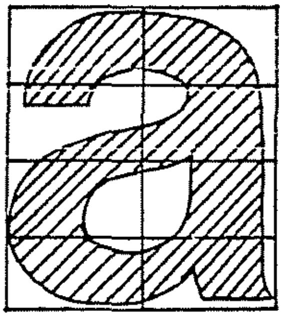
(c)

“*off*’ (*off*)

(d)

“*off*” (*off*)

Figure 3. A semi-image-space approach to collision avoidance in text setting



readability. This is, of course, a function of the human pattern-recognition mechanism, and likely has no rigid rules whatever, as I have already argued (in Section 2.5, "Counterexamples and 'Fully Automatic High Quality Music Notation' "). This makes the problem easier in some ways, but almost certainly harder overall. I will go into more detail on this question in Sec. 5.2.1.5.

Avoiding collisions of any type can best be thought of as having two totally independent components: *detecting* collisions in a proposed layout, and *resolving* them by moving one or more of the symbols involved. Literal collisions can be detected relatively easily. The problem of detecting them arises also in "hidden-surface" algorithms — algorithms used in displaying projections of three-dimensional scenes on conventional "flat" displays. These algorithms work by computing what parts of objects are hidden by other objects closer to the observer's eye and are therefore invisible. A tremendous amount of work has been devoted to such algorithms, and they are now rather well understood.[1] Identifying perceptual collisions, however, is not relevant to hidden-surface algorithms and is not nearly as well understood.

The second component of the problem — resolving collisions in proposed layouts — appears to be still less understood.[2] Which symbol or symbols should be moved, and where should it/they be moved to?[3] This subproblem apparently has never been attacked in anything like a general way for either type of collision. Not surprisingly, there are important differences in how collisions can be resolved, but the differences appear unrelated to the literal/perceptual dichotomy. For example, in circuit schematics, any component can be translated at least a little in any direction, and many can be rotated arbitrarily. In CMN, on the other hand, most symbols can be translated along only one axis, and none can be rotated at all except for some rather trivial cases (symbols that have radial symmetry and whose rotation is therefore invisible, certain note modifiers (Sec. 2.3.6.1) that can be rotated by 180 degrees); in this respect CMN is harder to deal with. These differences are substantial enough that none of the large amount of work on automatic placement in computer-aided design seems applicable to CMN. (For a discussion and bibliography, see [KIRK83].)

I will now discuss examples of both literal and perceptual collision avoidance, with attention to representation, then consider how all this applies to CMN.

5.2.1.1. One-Dimensional Collision Avoidance in Typesetting of Text

Some one-dimensional formatting problems related to collision avoidance occur in typesetting of text. Despite the simple parameters — only one dimension and literal, not perceptual, collisions — they are not completely trivial. Consider the spacing required to print the pairs of letters "AV" and "Wo" as opposed to, say, "AN" and "Wh": Fig. 2*a* shows all of these as set by TROFF [OSSA76, KERN78b] (more precisely, by ITROFF, the Imagen laser printer version of TROFF) in Knuth's Computer Modern font [KNUT82]. The problem here is what is called *kerning*. To make these look correct to the sensitive eye, most experts agree that one must take into account the shapes of the characters [BIEG76]: when "A" is followed by "N" it requires more room than when it is followed by "V", since in the latter case the two characters can share some space (Fig. 2*b*). "Kerning" is the term for this sharing of space, but the concept generalizes naturally to include the opposite situation, i.e., the situation where allocating each character the obvious amount of space results in consecutive characters being too close (Fig. 2*c*, again set by TROFF). Incidentally, note that the "ff" in Figure 2*c* (as well as in the text just now) is really a single character, called a *ligature*. In English, only a handful of ligatures are used, and they do not complicate kerning significantly.

Many typesetting systems have an automatic kerning capability; TROFF does not, although one can accomplish it by manually overriding TROFF's positioning. Automatic kerning is ordinarily done by table lookup on pairs of characters. A pure object-space (see Sec. 3.3.6) approach like this is hopeless for CMN and, in fact, does not work that well even for text, since it requires large, mostly empty, tables for use within each font. Worse, kerning is sometimes needed *across* fonts, as, for example, in Fig. 2*c*, where italic characters adjoin roman punctuation marks, and supplying a table for every possible pair of fonts would be a ridiculous overkill. Few if any text formatters attempt to kern across fonts automatically. TROFF again does not, and, for this dissertation, it had to be corrected manually in several instances (Fig. 2*d*). A new "semi-image-space" approach, recently developed by Electronic Information Technology, divides each character into eight zones as shown in Fig. 3. In most typesetting systems, the size of each character in a given font is described by a single number giving its overall width; there are kerning tables of the type I have described, but no explicit information about the *shapes* of the characters. In the EIT system, however, the size information for each character includes eight additional numbers, each of which says how much space on the outside of its horizontal "half slice" is vacant. This information, which essentially gives the shapes of the left and right edges of the character with very low resolution, is used in a straightforward way for kerning. For text, this is in several respects a better solution than kerning tables. It automatically works across fonts just as well as within them. It can easily be parameterized to suit a typographer's taste: varying a single number can produce any amount of kerning from none to enough to make the characters touch. (This is something like varying the size of the envelope in the literal near-collision model.) More important for our purposes, however, although this representation is one-dimensional, it does not appear hard to extend to the two-dimensional domain of CMN, for example by describing the top and bottom edges of the symbols as well as the left and right ones.

5.2.1.2. Perceptual Collision Avoidance in Cartography

A recent paper by Hirsch [HIRS82] describes an algorithm for "automatic name placement around point data". Hirsch says, "Three basic map features annotated by names are points, lines, and areas. The placement of names for these features is governed by the principle that the name and its object should be easily recognized." Hirsch is concerned solely with placing names on features of the first type so that they do not collide perceptually with other names or with other features of that type. He cites the opinions of various authorities on what relative positions of point and name are most desirable, and observes that, in the process of placing names around point data manually,

> three general procedural phases can be identified. First, a given set of names is selected from a gazetteer. Second, during a layout phase, names are iteratively placed in an attempt to find an overall solution. For instance, the cartographer might start out with several promising name configurations. As he proceeds, it may be necessary to rearrange some of the names already placed, choose a smaller letter size, or exclude some names. In this phase, the gen-

> eral procedure is to work on the lettering draft from an area of symbol density outwards placing the larger names first, then the smaller. Given an acceptable layout, the final phase is the actual placement.

Hirsch's algorithm assumes that names are always written horizontally. It is designed to place the names so that they do not overlap, and so that each clearly refers to its point symbol. It attempts to do this by (1) requiring that the distance from any labelled point to its name be a constant, in other words, that the "name" (presumably, the closest point of the name) lie on a predetermined circle around its point; and (2) not permitting any part of any other name to enter a point's circle. Thus, the major variable is the direction from each point to its name. In addition, if the direction is 0, 90, 180, or 270 degrees, the name can slide along a tangent to the circle at that angle. Hirsch begins by putting each name in its preferred position, then looking for overlaps. I could go into more detail, but it should be clear by now that this procedure is so specific to its domain as to be of no value at all for CMN. Hirsch references a thesis on automatically positioning feature names on maps, which might be more relevant, but I was unable to obtain a copy.

Hofstadter [HOFS83a] has suggested a perceptual collision that might occur on a map that would take significant intelligence to detect. If a hyphenated name were to appear in large type on a map, the hyphen might appear to underline another name in much smaller type.

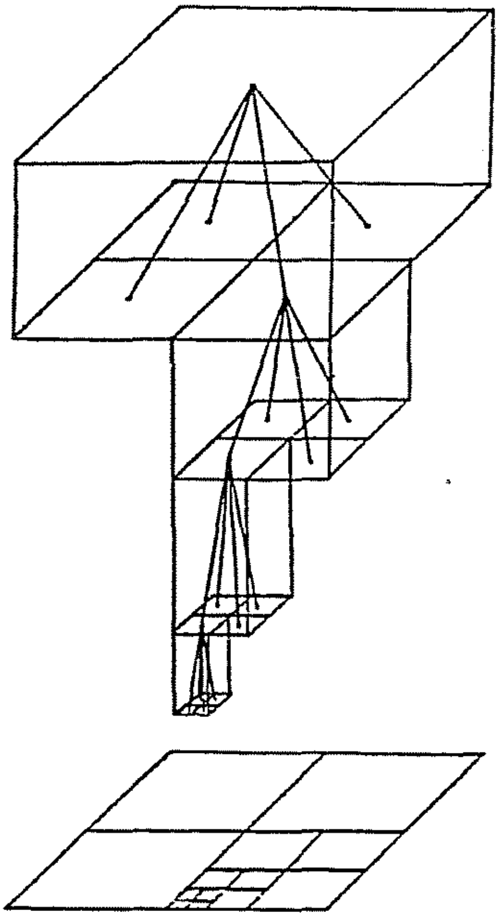
5.2.1.3. Representations for Collision Avoidance

Perhaps the most obvious representation of a page for collision avoidance is with a bit-matrix, simply indicating (with appropriate resolution) whether each "point" on the page is occupied or not. This seems especially appropriate with raster scan displays, since such displays already have available just this information (the resolution being, of course, one bit per pixel). However, in order to be able to decide what to do when a collision is detected, we really need to know what symbol each occupied point is part of. This is certainly feasible with some frame-buffer systems that have many bits per pixel.[4] To actually describe the image, we need only one bit per pixel (CMN is rarely printed with either color or gray scale!), so the remaining bits are free for the symbol descriptor.[5]

Unfortunately, the advantage of the bit-matrix data structure disappears on any nonraster device, and even on raster devices where the frame buffer is not "memory-mapped", that is, where it is not directly addressable by the processor. An approach that maintains device independence and also looks reasonably efficient is to keep the same information with a different data structure, namely the *quad tree*.

Quad trees were first used in the late 1960s, for example in a hidden-surface algorithm developed by Warnock [WARN69] (although the term "quad tree" is still not used universally). They are now used extensively in several fields of graphics, for example animation [HUNT78] and cartography [WEBB83]. Hunter studied the quad tree extensively; he defines one [HUNT78, p. 51] as "a tree whose nodes are either leaves or have four children." Thus, as the name suggests, they are quaternary trees. As used in computer graphics, quad trees recursively subdivide the (two-dimensional) picture in question into smaller and smaller rectangular quadrants, with leaves most commonly representing areas of uniform value (conventionally called "color", although it may be any information associated with a region) (Fig. 4). Thus, quad trees represent the picture with variable resolution: high in regions where it changes rapidly, low where it doesn't. As is intuitively obvious, they are efficient in terms of memory requirements for most applications; it is less obvious but equally true that they can be efficient in terms of computation time. Fast algorithms are known for many operations on quad trees, e.g., superimposing regions represented by them and finding neighboring regions [HUNT78], and for converting between them and such other representations as polygons [HUNT78], boundary codes [DYER80, SAME80], and rasters (i.e., bit-maps) [SAME81]. Incidentally, even with memory-mapped frame-buffer devices, quad trees may be preferable to direct use of the frame buffer, since they may improve efficiency in two ways: (1) by making it possible to determine very quickly whether or not a large region is free, and (2) by making the resolution the program deals with independent of the resolution of the hardware (this can improve efficiency because lower-than-hardware resolution will probably work quite well in most cases, and execution time for most algorithms is of course an increasing function of resolution).

Figure 4. A quad tree



Note that both bit-matrix and quad-tree representations are of the image-space variety (Sec. 3.3.6). It makes sense to consider such representations, since the problems we are trying to solve are purely on the visual level.

5.2.1.4. Perceptual Collisions and Artificial Intelligence

As early as Sec. 1.3.1 of this dissertation, I expressed the opinion that practical results in computer setting of music could be achieved in the near future, but only by avoiding the AI techniques which, as I argued in Sec. 2.5, are vital for "Fully Automatic High Quality Music Notation". Nowhere is this more true than in avoiding perceptual collisions.

As I said in Sec. 2.5, making decisions in music setting can involve both common sense and the (local and global) semantics of the music to an arbitrary extent. I cited there a number of examples of literal collisions in music from well-known publishers, as well as some things that are more relevant to the current discussion: examples of symbols interrupted to avoid collisions and of symbols written with nonstandard shapes to avoid collisions. How could a system (whether human or computer) decide to do these things? Obviously, only by comparing the possible alternatives — *all* of which, in many cases, violate the explicit conventions of CMN in some way! — and deciding what can most easily "slip", to use Hofstadter's term. Hofstadter argues that such decisions are the key element of human intelligence [HOFS83b]:

> When faced in life with a complex decision to make, with pressures being exerted on us in various directions, often we find that something in our mind "gives", in the sense of yielding under pressure. We do not consciously choose what will give; rather, we discover, to our surprise, that something simply *has* given. In many such situations the intensity of the pressures is small enough that the decision is not crucial; in them, the same kind of yielding or giving takes place, but because the stakes are less, we often do not notice, either during the decision or after it, that there was a conflict to resolve or that it was resolved by something's yielding.
>
> These kinds of situations exemplify what I call mental "slipping": the conversion of one concept into another related one under pressure . . . slippage is *the* mechanism, not just *a* mechanism, by which thought takes place in time in a

> mind/brain. The determination of what actually slips is the result of the differential *slippabilities* of the various facets of a mental representation of a situation. These slippabilities are not explicit *numbers*, but qualities dependent on the entire interwoven structure, and they emerge only when the entire cognitive structure is stressed from the outside, *i.e.*, pushed or pulled or otherwise disturbed, and forced to yield.

See also [HOFS79, pp. 633 – 40, 641 – 80], and [HOFS82b].

5.2.1.5. Avoiding the Entire Problem of Collision Avoidance

The cases we have been considering suggest the hypothesis that the perceptual-collision avoidance problem is so hard that workers have completely avoided *the problem itself.* The few attacks that have been made on it have been so dependent on domain-specific knowledge that it is hard to see how they could be adapted to other areas. In music setting, I am not aware of anyone thus far who has failed to avoid the problem; that is, everyone (to my knowledge) has sidestepped it. (Additional evidence of the difficulty of the problem is given by my Counterexamples (Sec. 2.5). It is obvious in many of them that the need to avoid collisions is what drove the composer or editor to employ idiosyncratic notation.)

SMUT exemplifies one way of avoiding the problem: it does not even have a mechanism for *detecting* collisions, whether perceptual or literal, but it nonetheless attempts to prevent them, simply by following rules of thumb for the placement of various symbols. For example, groupet auxiliary numerals are positioned vertically just outside the range occupied by noteheads and stems of the notes in the group, so that they cannot collide with the noteheads or stems. However, this does not guarantee that an auxiliary numeral will not collide with something else, say a slur or character string; it may actually *cause* such a collision by positioning the numeral further away from the notes than it needs to be. In fact, SMUT as currently implemented *cannot* look for and avoid collisions, because it has no "mental image" — no image-space representation — of what the page it is constructing looks like. I have already discussed (in Sec. 5.2.1.3) how such an image might be maintained, both in terms of data structures and in terms of the information contained in them.

5.2.1.6. Collision Resolution in CMN

Suppose now that we have an algorithm for detecting some type of collision in CMN. Naturally, we much prefer to handle perceptual collisions, but that is rather a tall order, and (as a first approximation) the ability to deal with literal collisions would probably be useful. Certainly part of the representation involved would be information about the shapes of the symbols, as suggested in Sec. 5.2.1.1. In any case, what domain-specific knowledge is available to help us decide how to resolve collisions, and how should it be used? Of obvious importance are the permissible translations (in the geometric sense) of CMN symbols. But we really need to know, not just whether it is permissible to move a clef horizontally or vertically, but — assuming it collides with a character string — whether it is more desirable to move the clef or the string. A first attempt at giving this information appears in Table 1. The larger the number a type of symbol has in the "Vertical slippability" column, the less undesirable it is to move a symbol of that type in that axis. "0" means the symbol type has no flexibility and should not be moved in that axis, no matter what the circumstances. The same holds for the "Horizontal slippability" column. The numbers given are rough guesses and could undoubtedly be refined.

With this information, a basic collision-resolution procedure, on which many variations and improvements are possible, is obvious. (For example, one might have separate numbers for up and down motion instead of a single vertical motion number, and similarly for left/right motion instead of horizontal. Or one might define the numbers in the manner of TEX "glue" as in some way limiting the distance the symbol can move, allowing larger displacements but with an appropriate penalty scheme to discourage them.) At the same time, I hope it is also obvious, considering all I have said, that the question of "slippability" is an extremely deep one that cannot possibly be satisfied in general by comparing a few numbers: shallow collision avoidance schemes like this one may certainly work better than none at all, but they have no hope of solving *all* the possible sets of conflicting requirements in CMN.

TABLE 1

Item	Vertical slippability	Horizontal slippability	variable aspects
Noteheads, stems	0	6	
Augmentation dots	2	7	
Accidentals	0	4	
Rests	6	5	
Note modifiers	4		
Beams (affect stem endpoints)			LO
Slur endpoints	8	3	S
Tie endpoints	1	2	L
Groupet accessory numerals and brackets	5	1	LO
Character strings	9	9	
Barlines and repeat signs	0		
Clefs	0	8	
Key signatures	0		
Time signatures	0		
Grace notes	0		
Octave signs	7		L
Glissandi	0		LO
Pauses	3		

Codes for "Variable aspects": L: length; O: orientation; S: shape.

5.2.2. Symbol Placement by Simulated Annealing

A recent paper by Kirkpatrick et al [KIRK83] presents analogies between statistical mechanics and combinatorial optimization that appear to me very promising for placing symbols in music notation. The paper's summary says

> There is a deep and useful connection between statistical mechanics (the behavior of systems with many degrees of freedom in thermal equilibrium at a finite temperature) and multivariate or combinatorial optimization (finding the minimum of a given function depending on many parameters). A detailed analogy with annealing in solids provides a framework for optimization of the properties of very large and complex systems.

Annealing is the process of altering the physical structure of a solid by "heating [it] to and holding [it] at a suitable temperature and then cooling [it] at a suitable rate for such purposes as reducing hardness, improving machinability, facilitating cold working, producing a desired microstructure, or obtaining desired mechanical, physical, or other properties." [AMER61] More specifically, the type of annealing to which Kirkpatrick et al refer involves melting the solid, then lowering its temperature slowly, spending a long time at temperatures in the vicinity of the freezing point. The result of the annealing may be thought of as minimizing the value of some function of the object. In *simulated* annealing, computations are done on some datum or set of data in order to minimize a given function of the data, usually called the "cost function". Clearly, the process also involves analogs of temperature and of the concepts of melting and freezing. In our case, the printed music is the solid; the amount of freedom the symbols on the page have to move around independently is temperature, and melting means changing from no such freedom to some freedom; and the cost function is one that describes numerically the "badness" of the particular arrangement of symbols on the page. Obviously, an important ingredient of such a function will be an estimate of the seriousness of any collisions involved — not an easy matter, as we have seen. However, given a badness measure, "simulated annealing" looks very promising as a technique for improvement. Referring to attacks on NP-complete problems, which frequently require prohibitive computational effort for exact solutions, Kirkpatrick et al write:

> There are two basic strategies for heuristics: "divide-and-conquer" and iterative improvement. In the first, one divides the problem into subproblems of manage-

> able size, then solves the subproblems. The solutions to the subproblems must then be patched back together . . . In iterative improvement, one starts with the system in a known configuration. A standard rearrangement operation is applied to all parts of the system in turn, until a rearranged configuration that improves the cost function is discovered. The rearranged configuration then becomes the new configuration of the system, and the process is continued until no further improvements can be found. Iterative improvement consists of a search in this coordinate space for rearrangement steps which lead downward. Since this search usually gets stuck in a local but not a global optimum, it is customary to carry out the process several times, starting from different randomly generated configurations, and save the best result.

(Although Kirkpatrick et al do not say so, presumably the process is always restricted to syntactically valid configurations; there would be no point to allowing invalid configurations. Also, in setting music, where the cost function is tremendously complex, it would be sensible to start, not with any random legal configuration of the symbols, but rather with a configuration generated by methods like SMUT's.) The essential difference between iterative improvement and simulated annealing is that the former is deterministic while the latter is not. In both processes, a step (that is, a rearrangement) that improves the cost function is always accepted. In annealing, however, a rearrangement that *hurts* the cost function may still be accepted, depending on a random variable. The rationale for this is to avoid getting stuck in a local optimum, to accept local worsening in hope of global improvement, to allow breaking "logjams". The annealing process accepts or rejects steps with the following procedure: let ΔC be the change in the cost function produced by that step and let T be the current temperature. Then if $\Delta C \leq 0$ the step is always accepted, while if $\Delta C > 0$ the probability of accepting it is $\exp(-\Delta C/T)$. Thus the probability of a locally harmful step being accepted decreases as the amount of harm increases, but it increases as the temperature increases. Note particularly that at zero temperature no harmful changes are accepted. The effect might be thought of, in Hofstadter's terminology, as parallel slipping of many things, using some randomness to help decide what slips are accepted. Kirkpatrick et al argue convincingly for the superiority of annealing:

> Annealing . . . differs from iterative improvement in that the procedure need not get stuck since transitions out of a local optimum are always possible at nonzero

temperature. A second and more important feature is that a sort of adaptive divide-and-conquer occurs. Gross features of the eventual state of the system appear at higher temperatures; fine details develop at lower temperatures.

5.2.3. Texture and the Importance of Stupidity

Severo Ornstein, in speaking of his and John Maxwell's Mockingbird system, made the thought-provoking comment [ORNS82]: "We were very careful to build a 'stupid' system first into which 'smartness' could be added later. That allowed us to finish — and handle (piano) scores of more or less arbitrary complexity." This view of stupidity as desirable because it promotes flexibility is, of course, much different from the usual one that a system should be as smart as possible so as to relieve its users from the need to specify low-level details of no interest to them. In fact, however, these two viewpoints dovetail nicely. The resolution lies, first, in the observation that Ornstein's statement is really emphasizing the desirability of modularity of levels of control, and second, in the attitude I express in [BYRD80] and Sec. 5.3 below that a complex system should let users work at any level *and* change levels very easily.

Seen from Ornstein's perspective, SMUT suffers from a severe case, not of over-smartness, but of overconfidence in its smartness: its high-level machinery depends heavily on certain assumptions about CMN that are incorrect for a great deal of music — perhaps most. The most important way in which this is true has to do with texture (see Sec. 2.3.6) and its effect on symbol placement.

SMUT has built into it the assumptions that each staff will have a constant number of voices, that that number is no more than two, and that each voice belongs to a single staff.[6] As I have suggested before (e.g., Sec. 1.4), all three of these assumptions are violated constantly in keyboard music, and occasionally in music for other instruments. The number of voices on a typical staff of keyboard music varies continually, often reaching three, and voices frequently cross from one staff to another. On the other hand, SMUT's low-level machinery could profitably be used on virtually all CMN if only it could be accessed without interference from the high-level machinery.[7]

SMUT is not the only music-setting system with limited understanding of texture; in fact, I know of no system that attempts to handle *automatically* more than two voices per staff, a varying number of voices per staff, or voices shared between staves. There is good reason for this: as I pointed out in Sec. 2.3.6.2 and 2.4, multiple voices per staff and interdependence between staves lead to great complexity. This emphasizes again the desirability of easy level switching.

It is probably safe to say that the major unsolved problems of automatic music setting are the two aspects of symbol placement we've just discussed, namely collision avoidance and handling of complex textures. Neither is likely to be solved in a very satisfactory way any time soon. This is especially true because the two are closely interrelated: complex textures are much more likely to lead to problems with collisions than are simple ones. See Sec. 5.6.

5.2.4. Modularity of Knowledge and Single-Character Placement

The issue of modularity of knowledge relates to some symbol-placement problems that are much simpler than those of collision avoidance. A very common problem in all kinds of formatting systems is that knowledge of how much space various symbols occupy is stored in more than one place, so that it is possible for different parts of the system to behave inconsistently. TROFF, for one, appears to suffer from this problem. The version of TROFF available to me when I began working on this dissertation had problems spacing text properly, but set the mathematics of Figs. 46 and 47 of Chapter 2 correctly. A later version did better with text, but much worse on those figures. A few instances of improper spacing of text still show up in this dissertation: for example, look at the "isio" in the word "collision" in any of the section headings of this chapter. TeX also suffers from the problem, as does SMUT. The problem is a curious one in that, when symbols are drawn by software, it is (in principle, at least) easy to solve; when they are drawn by hardware, it is more or less unsolvable, since no part of the software really knows how much space the symbols need, but is also less likely to cause trouble, since symbol space requirements are likely to change far less often.

Note, incidentally, the relationship of this problem to kerning and to collision avoidance: this is a local, one-symbol-at-a-time problem, while kerning involves two symbols at a time, ligatures (in English) involve two or three, and collision avoidance — at least in CMN — any number. But this problem is also basic to kerning and collision avoidance in that if a system does not have accurate knowledge of the space requirements of individual symbols, it cannot possibly handle other symbol placement problems.

Clearly, even if the symbols are drawn by software, the only routine that really knows how much space a symbol occupies is the one that draws it. In SMUT, all of the symbol-drawing routines are in Pass IV; but in order to do punctuation, Pass III needs to know how much space the symbols occupy, at least horizontally (the current implementation concerns itself with vertical space requirements only in a very crude fashion). Subroutine CALCXY answers questions from JUSTIF about horizontal space requirements. In addition, there are a few cases where Pass IV routines do small-scale, local, vertical positioning and therefore need to know how much vertical room other Pass IV routines will need for their symbols.

Currently, routines in SMUT have built into them assumptions about the space requirements of symbols drawn by other routines, mostly via DATA statements. A much better way to handle this would be to *ask* a symbol-drawing routine how much space a given symbol will occupy. This could be done simply by adding two parameters to the calling sequence of each such routine, one input to specify "print" or "return information", and one output in which to return the spacing information.

5.3. EDITING AND FORMATTING, DESIGN AND DRAFTING

The work of preparing visual information in many disciplines, including music, for presentation to others is commonly divided into "editing" and "formatting". Two recent surveys of editing and formatting, respectively, that appeared in a single journal issue are those of Meyrowitz and Van Dam [MEYR82] and Furuta et al [FURU82]. Each paper mentions work on various types of information but concentrates on text[8], and each acknowledges the relevance of the other area. Furuta et al define two types of hierarchies of objects, one composed of "abstract" objects (in text, items like characters, "logical" words which might appear in dictionaries, paragraphs, sections, headers) and one of "concrete" objects (in text, items like representations of characters in a particular font, two-dimensional words with possible hyphenation, and lines). "A *document* is an object composed of a hierarchy of more primitive objects", presumably (though they do not say so) either all abstract or all concrete. In this terminology, editing operations are mappings from abstract objects to abstract objects or concrete objects to concrete objects. Formatting operations are mappings from abstract objects to concrete objects. Meyrowitz and Van Dam take the interesting position that formatting is really an integral part of every editor, and suggest that multiple formats are becoming more important.

Steven Johnson has pointed out [JOHN81] a more abstract and general way of looking at the process, namely in terms of what the user is trying to accomplish, not what she or he is doing to accomplish it. In this view, the dichotomy is not "editing" versus "formatting", but "design" versus "drafting". (In the other view, it is hard to classify input as either editing or formatting; here, it is clearly part of design.) Computer processing of music is one of a class of problems that might be called "computer-aided design" in a much broader sense than the usual one. The phrase is ordinarily applied to the automation of the fields that themselves go under the name "design" — electronic and mechanical design, for example; but computer processing of music and text, among other things, has many similarities to computer-aided design. This terminology, again, is used only in the "design" fields. In music, design is called "composing", and drafting is called "copying" (or "engraving" or "autography", etc.); wandering slightly outside of graphics into text processing, we find the terms are "writing" (supported by text editors) and "formatting" or "setting". Whatever terms are used, the essence of the distinction is that design, in the strict sense, is concerned primarily with semantics and secondarily with syntax. Drafting, on the other hand, is ideally concerned only with graphics and possibly syntax (see Sec. 2.4 for terminology) and should under no circumstances alter the semantics of the material. Thinking more concretely reveals the usefulness of considering the process as having a third part, what is usually called "editing" but might be better termed "design revision". The reason for this distinction is, of course, that in most real-life situations the design and drafting tasks do not occur strictly sequentially, but instead one switches back and forth between them more or less frequently. Ideally there should be no overhead in the switching, so that one can do it as often as desired without incurring any penalty. (This switching has occurred in music composition to a very limited extent simply because the nonautomated tools hitherto available have imposed so much overhead. This should be evident to anyone who has looked at a typical manuscript of Beethoven, with its multiple scratched-out versions of a single passage.) In any case, my research has been concerned solely with the drafting subproblem; however, it is important to realize that an integrated system that deals with both design and drafting will nearly always be more useful than two separate systems because it will greatly facilitate the switching.

In fact, in most fields, two types of systems have traditionally been built, but the types really correspond to a "batch/interactive" dichotomy, not to the "design/drafting" dichotomy. In a discussion of computer-aided design for generating integrated circuit mask layout data, Trimberger comments [TRIM81]:

> Two primary methods for generating integrated circuit mask layout data are *interactive graphics* and *layout languages*. Each has tasks which it does well and those which it does not . . . Often the limiting factor in the speed of design is the time it takes to plot the data. Interactive graphics systems provide "instant plotting", enabling the designer to iterate extremely quickly on the design. Interactive graphics systems also provide a powerful "language" for handling the data. For example, the user may point to the object of his attention or to a desired position, rather than search for certain numbers in a program printout or type numbers . . . But interactive graphics systems do not allow . . . looping constructs [except some that are] severely limited, conditional geometry or relative positioning. Much more powerful language-style operations are needed. Layout languages attempt to resolve these problems . . . Unfortunately, languages specify graphic positions in an awkward fashion, by numbers . . . Current languages force the user to go through a tedious and time-consuming edit-compile-plot cycle. Interactive techniques have attempted to get rid of this lengthy cycle, but have been usually aimed only at the graphic form and not at the language form.

With minor rewording, these comments apply almost equally well to text processing or to music editing and formatting. The analogy to music is slightly better because it usually has more repetition than text.

Even if the design phase is done manually, a computer must obviously have the resulting design fed into it before any drafting can be done. In most fields, certainly including music, this is so far from trivial that substantial editing is usually required to correct errors in the input process. Of course, substantial editing can be done without a good editor — in particular without one that is good for the problem domain. But having such an editor certainly helps, and if one is available, it is hard not to think of doing the design work on-line.

5.3.1. Supporting Multiple Levels of Control

In the paper just cited, Trimberger goes on to discuss his Smalltalk-based "Sam" system, which attempts to combine the two approaches by maintaining a data base that is independent of both the graphics and the language form, and from which the two forms are generated, displayed on the user's terminal in separate windows, and continuously updated.

> The left side [of the display] shows the program view of the design under edit, the right side shows the graphics view. The user may move the viewing location in either window and may make edits to the data in either window. When the design is changed in either window, the change is reflected immediately in both windows. The data displayed in the windows are *pictures* of the data structure. The data structure is the base form . . . When the user makes what appears to be a modification of the data in either window, the commands are translated into calls on procedures in the data structure to carry out the action. The data structure makes the modification and causes both displays to be updated . . . The data structure . . . is more than a conventional design automation database, consisting as it does of objects which have both data and code attributes.

Trimberger discusses two problems — "expression update" and "iteration update" — that arise because his system allows changes to be made "in two different forms which must remain consistent." Both of the problems occur when the user has described something at a high level of description, i.e., algorithmically, and now modifies an instance of it at a low level, i.e., graphically, or one might better say, "manually". This sort of level changing, and either problem, could occur in editing a piece of music notation. So can the reverse kind of level change, going from a low level to a high one, as described in the following excerpt from [BYRD80]:

> Now let's say that a user formats his/her music; then, in order to save vertical space between staves, makes a low-level change: "move the left end of this beam 5 units right". (Music engravers actually do this sort of thing.) Now the user discovers a whole measure missing and inserts it — a much higher-level change which requires completely repositioning the beam. To do the right thing with it, the program will need to understand why the user moved the beam to begin with.

Other problems of going from a low level to a higher one are legion: in the typesetting world, for example, doing any fancy "manual" alignment at all, then either changing the wording and reformatting, or switching fonts (and therefore character widths). In the latter case, both changes are made graphically, but there is still a change in level of description: now, something has been described at a moderate level and "manually" modified at a low level, and the user then wants to specify a change at a high level in the same "neighborhood". This question of switching levels is crucial.

Working at a low level must be allowed so that users can "manually" do things that the system does not understand how to do. This is clearly of great relevance to CMN not only because, as I attempted to show in Secs. 2.5 and 5.2.1.4, no program will be able to handle CMN totally satisfactorily until the entire problem of AI is solved, but more importantly because for a long time to come, even many less difficult problems of CMN will be more easily settled by human intervention than by automatic methods.[9] Switching levels should be easy because situations like the one I just described (the example of "going from a low level to a high one") will presumably be quite local in any given piece of work but may occur in it many times; as a result, the user will want to work at a high level most of the time, and switch to a lower level for brief periods but frequently.

A solution in the text domain to some of the problems of supporting both high and low levels of control, in many ways similar to Trimberger's approach, is found in the Xerox STAR [SEYB81, MEYR82]. A basic problem with the common "what-you-see-is-what-you-get" text formatting systems (e.g., WORDSTAR) is that, once they have finished a formatting operation, they do not retain any memory of the operation's parameters. Although the system supports high- as well as low-level operations on the text, it only retains very low-level information, namely the current formatted version of the text. In contrast, the STAR text setting system maintains for each document one or more "property sheets" for each of several entities: characters, paragraphs, page layout, etc. For example, the paragraph property sheet includes such information as alignment (flush left, flush right, or centered), whether to hyphenate, margins, and so on. Each type of property sheet can change at any point in the document. This provides a record that describes the formatting of the text and that can itself be edited. The record is conceptually high level, but, since it can change arbitrarily frequently, any of the properties can apply to a very small amount of the text, perhaps only a single character.

5.4. AUTOMATING WHAT HAS ALWAYS BEEN DONE MANUALLY

As Knuth has shown [KNUT79, KNUT81], automatic text formatting by computer is now advanced enough to give results superior to manual in some ways, and equal to manual in most. Not only is formatting of music inherently far more difficult than that of text (as we saw in Sec. 2.4), but far less effort has been expended on it. As a result, the best computer-set music — and by "computer-set" I mean here *fully automatically computer-set* in the sense of Sec. 1.3.1 — is still clearly inferior to the best manually set music in most ways. However, this does *not* mean that we should proceed by slavishly trying to imitate manual techniques in every case.

An example of this mistake, it seems to me, is Gomberg's general attitude towards music engraving, and specifically his algorithm for determining "ideal" horizontal space requirements [GOMB75, pp. 42 – 69]. It is far more complex than it needs to be because he attempts to use the same technique engravers use. On the other hand, sometimes it really does pay to imitate manual techniques if this is done properly, where the meaning of the word "properly" is not easy to specify. In [KNUT79] Knuth discusses a similar question about the production of digital versions of typefaces, which several people had approached as the purely technical question of making good digitized copies (p. 17ff):[10]

> I felt that the whole idea of making a copy [of an existing font in digital form] was not penetrating to the heart of the problem. It reminded me of the anecdote I had once heard about slide rules in Japan. According to this story, the first slide rule brought to the Orient had a black speck of dirt on it; so for many years all Japanese slide rules had a useless black spot in the same position! The story is probably apocryphal, but the point is that we should copy the substance rather than the form. I felt that the right question to ask would not be "How should this font of type be copied?" but rather: "If the great type designers of the past were alive today, how would they design fonts for the new equipment?"

Similarly, in [KNUT82] Knuth describes the way his METAFONT system draws characters by simulating a pen whose nib's size and shape are variable. This technique does not imitate modern design of characters for phototypesetting or, formerly, printing with movable type; rather it imitates the technique used by the calligraphers from whose work modern letterforms evolved.

Gomberg was certainly aware of this issue of how tasks should be automated, as shown by his observation (p. 30) that

> There are essentially two ways one can attempt to accomplish the activity of a human being with the aid of a computer. One is to duplicate as nearly as possible every step of the human activity ... The second approach ... is to attempt to use the same input as the human being and, by whatever means possible and convenient, produce the same output as the human being.

He then gives several reasons why a program cannot really duplicate the behavior of a human music engraver:

> [The engraver] has eyes with which he can quickly scan for important features and ignore those that his experience tells him are trivial. He can easily adjust and readjust his field of concentration, if necessary, from to one to several pages. He uses short-term memory, but he can refresh it with the important details literally "at a glance". Thus he is able to make gross level judgements about whether there are special problems that need to be considered in some detail before proceeding further, or whether the current pages ... are quite ordinary.

Therefore, Gomberg says, "alternative means must be found."

5.5. PUNCTUATION AND CASTING OFF

In punctuation and casting off, music-setting systems have much to learn from typesetting systems, in particular TEX's "glue", "boxes", and "penalties". The whole-movement-at-a-time method for casting off that Smith uses in MSS also looks excellent, if only because of the peculiar tradition that a piece of music completely fills its last page (Sec. 2.3.6.4). Unless one attempts automatic page-turn decisions (an attempt that would be likely to fail: again see Sec. 2.3.6.4), the constraints involved in the punctuation process are simpler than those in collision avoidance.

5.6. RESEARCH PROBLEMS

Here is a brief list of some of the remaining research problems in computer music setting, nearly all of which I have already discussed. As I have tried to make clear in this chapter, the first two problems are of great difficulty and great importance, both practical and theoretical, for many fields, not just music setting. Most of the others, though very difficult to solve fully, are of relatively slight practical importance, since they can be handled rather easily by interactively correcting rough algorithmic solutions.

(1) Automatically placing symbols to avoid perceptual collisions and maximize readability, especially in complex textures. This problem appears to be "AI-complete" (see Chapter 2, note 47).

(2) Supporting multiple levels of control in a really convenient way.

(3) Deciding automatically where to put page turns, in scores as well as parts.

(4) Deciding automatically what cue notes to use, if any, after long rests in one performer's part.

(5) Deciding automatically on clef changes.

(6) Deciding automatically what groups of notes to beam together.

(7) Choosing cautionary accidentals automatically.

The various input methods bring in a host of additional problems, which are discussed in Chapter 3.

5.7. SOME CONCLUDING SPECULATION: PROSPECTS FOR COMPUTER-SET MUSIC

Using as an example Elliott Carter's exceptionally complex *Double Concerto for Harpsichord and Piano with Two Chamber Orchestras*, engraved in about 1962, Gomberg speculates rather pessimistically about the future of computer music setting [GOMB75, p. 89ff]:

> The estimate that I have received from people in the industry is that probably no page of the Carter was set for less than fifty dollars, and that [some] pages . . . might well have cost in excess of one hundred dollars. It is probably reasonable to estimate that these figures would be doubled today . . . I have tried to project very carefully the cost of producing an "average" page of the Carter Double Concerto by means of the automatic processes described in this paper . . . After first making this estimate, which includes all labor and computer-related costs, I doubled it in order to allow for error and personal prejudice, and arrived at a figure of about twenty-five dollars a page . . . it is not possible to say that for music "half as difficult as the Carter" the cost would be reduced by half, but it is certainly true that one can expect to compete more than favorably with existing processes . . . Despite such a promising cost projection, the major obstacle to the culmination of this project as a commercially viable venture is lack of funding.

Gomberg estimates total development cost to complete his project (which, it will be recalled, was intended to be as automatic as possible) at $500,000, and suggests that the music publishing industry will be unable, and foundations and venture capitalists will be unwilling, to supply these funds.

Prospects for the completion of similar work now are very much better. For one thing, computer resources are, of course, much cheaper and more readily available than they were in 1975. For another, several workers, including myself, have already put a large amount of effort into computer setting of music, mostly without external funding. In fact, to my knowledge there are now three systems in regular use for music publishing (the Dataland Scan-Note system, Watkins's Musigraph, and Armando Dal Molin's), and several companies that make digital synthesizers have announced computer music-setting facilities of some sort (see Sec. 3.4 for details). (In 1975 there were no commercially available digital synthesizers, much less music-setting facilities for them.) No existing system comes close to Gomberg's goals of putting every jot and tittle in place automatically, but there can hardly be any doubt that systems will continue to improve, and — considering the high level of interest now evident — probably at a rapid pace. As I have tried to show, there is a fundamental and clear-cut tradeoff: if one insists on "Fully Automatic High-Quality Music Notation", one should expect to wait until most of the major problems of artificial intelligence are solved (not likely within my lifetime); but if one is willing to accept practical compromises, a decade should more than suffice.

NOTES

[1] In fact a paper comparing no less than ten such algorithms appeared some years ago [SUTH74]. Both of the standard texts on computer graphics, [FOLE82] and [NEWM79], include extensive discussions of hidden-surface algorithms (as well as hidden-line algorithms, which are closely related).

[2] Neither [FOLE82] nor [NEWM79] even mentions this problem. This is perhaps understandable, since they are dealing with *interactive* computer graphics, and collisions can always be avoided by interactive methods, without requiring any knowledge of collisions on the program's part. However, I have not seen a general discussion of collision avoidance anywhere else.

[3] Or should changes be made that might, as far as a program knows, change the semantics of the material? This is not at all unheard of in manual setting of CMN (or, for that matter, text,

where it implies changing the wording); in fact I know of one instance [HASS83] in which a composer made a change that unquestionably affected semantics, specifically rhythm, in order to avoid a collision. However, any such behavior obviously requires a *great* amount of intelligence, so I will ignore the possibility.

[4] In particular, it should be doable with any z-buffer system, substituting symbol descriptors for z-coordinates.

[5] In fact, for various reasons, music is occasionally printed with two or three colors, and for the same reasons plus feedback it might well be useful for a computer system to use colors. This could be accommodated simply by allowing two bits per pixel instead of one. See Chapter 3, note 16.

[6] As I pointed out in Sec. 2.3.6, the term "voice" is ambiguous. The meaning I intend here allows "thickening".

[7] SMUT actually does have a mechanism to do this, but only to a very limited extent. Its SHARE command dynamically tells SMUT to behave as if the current voice were the upper voice of a shared staff, the lower voice of a shared staff, or on an unshared staff. See Sec. 4.3.

[8] Meyrowitz and Van Dam make the statement that "graphics editors are described in [NEWM79] and [FOLE82]". This is, unfortunately, not true: [NEWM79] and [FOLE82] only describe general graphics principles, which could of course be applied to build a graphics editor. I know of no published survey of graphics editors.

[9] Programming languages provide many good examples of the importance of providing various levels of control. One is the well-known weakness of PASCAL for systems programming because it does not allow access to the low-level aspects of conventional machines. In my opinion, C has been more successful than PASCAL mostly because of its wider range of levels of control: it goes equally high in level and much lower.

[10] The same question might be asked about present-day electronic musical instruments, several of which go to great lengths to imitate the timbres of acoustic instruments. The analogy is an imperfect one, however: there have been type designers for centuries, but only in the last few years has technology approached the point where someone could possibly act as a "timbre designer" without spending most of their time on side issues involving (for example) the physics of cylindrical tubes.

"I think I should understand that better," Alice said very politely, "if I had it written down: but I can't quite follow it as you say it." "That's nothing to what I could say if I chose," the Duchess replied, in a pleased tone.

Lewis Carroll, *Alice in Wonderland*, Chapter IX

The Red Queen shook her head. "You may call it 'nonsense' if you like," she said, "but *I've* heard nonsense, compared with which that would be as sensible as a dictionary!"

Lewis Carroll, *Through the Looking Glass*, Chapter II

Postscript

The year since this dissertation was written has seen extensive work by several persons on computer notation of music and related topics. All I can do here is make a few brief comments and point to recent publications.

Jim Miller of Electronic Screen Productions has developed a highly interactive system that performs a number of functions in addition to CMN editing and output. His "Personal Composer" runs on an IBM Personal Computer with any of several commercially available digital synthesizers connected via MIDI (the Musical Instrument Digital Interface, a recently-developed serial interface standard) for real-time clavier input and audio output. The notation, produced on a dot-matrix printer, is nowhere near engraving quality but is quite readable. "Personal Composer" is scheduled for distribution in the near future by Yamaha Corporation. See [FREF83].

Passport Designs' new "Polywriter" (see the company's advertising literature [PASS84]) runs on an Apple II computer and produces output on a small dot-matrix printer. It accepts input, apparently in real time, from Passport Designs' own "Soundchaser" or via MIDI. It can handle 16 staves per page, and apparently up to 12 notes on a single stem per staff. Quality of the notation is poor, and Polywriter has some severe limitations, e.g., no durations shorter than triplet 16ths and very little flexibility in page layout. On the other hand, it can transpose, and it knows some of the notational properties of instruments, specifically position in an orchestral score and transposition. In the words of Chris Albano of Passport Designs [PASS84], "Polywriter is the most significant development since the music typewriter."

Syntauri Corporation is marketing still another program (called "Composer's Assistant") that takes real-time clavier input and prints it in CMN on a dot-matrix printer. Syntauri makes an inexpensive synthesizer that is driven by an Apple II computer, the computer on which Composer's Assistant runs. However, a recent review of the "alphaSyntauri" music system [AIKI83] says, "In its current form . . . this program still has some fairly severe limitations . . . What [it] needs is some editing software that would allow you to interact with your score while it's still on the screen . . . "

I only recently became aware of a system in use at the University of Illinois' Computer-based Education Research Laboratory (CERL). Carla Scaletti wrote of it [SCAL84]: "The CERL Music Group currently supports a music print program, written by Lippold and Dorothea Haken, which is heavily used by both musicologists and composers at the University of Illinois, and which allows for graphic or alphanumeric editing and the audition of entered scores."

The Xerox PARC Mockingbird system has now been described in an article in a widely-available publication [MAXW84].

Turning to a more theoretical area, in Sec. 5.2 I mentioned constraint systems as a promising approach to the symbol placement problem. A recent paper by David Levitt [LEVI84] discusses applications of such systems to music, although not specifically to music notation.

Bibliography

AIKI83 Jim Aikin, "Keyboard Report: The Alpha Syntauri", *Keyboard* 9,6 (June 1983), pp. 78 – 90.

AMER61 American Society for Metals, *Metals Handbook*, vol. 1, 8th ed. (1961).

ANDE79 Knud Dalbøge Andersen, letter to Orion Crawford, 14 May 1979.

APEL69 Willi Apel, ed., *The Harvard Dictionary of Music*, 2nd ed. (Harvard University Press, 1969).

BACK69 John Backus, *The Acoustical Foundations of Music* (Norton, 1969).

BARH60 Y. Bar-Hillel, "A Demonstration of the Nonfeasibility of Fully Automatic High-Quality Translation", Appendix III to "The Present Status of Automatic Translation of Languages", in *Advances in Computers*, Vol. I (F.L. Alt, ed.) (Academic Press, 1960), pp. 158 – 63.

BAUE70 Stefan Bauer-Mengelberg, "The Ford-Columbia Input Language" (in [BROO70]).

BIEG76 J.I. Biegeleisen, *Art Director's Workbook of Typefaces*, 3rd ed. (Arco, 1976).

BODO83 Lawrence Bodony, personal communication, May 1983.

BOKE72 Norbert Böker-Heil, "Plotting Conventional Music Notation", *Journal of Music Theory* 16, double issue 1 and 2 (1972), pp. 72 – 101.

BROO70 Barry S. Brook, ed., *Musicology and the Computer* (CUNY Press, 1970).

BUCH78 Alexander Buchner, *Mechanical Musical Instruments*, Iris Urwin, translator (Westport, Conn.: Greenwood Press, 1978).

BYRD74 Donald Byrd, "A System for Music Printing by Computer", *Computers and the Humanities* 8,3 (1974).

BYRD77a Donald Byrd, "An Integrated Computer Music Software System", *Computer Music Journal* 1,2 (1977).

BYRD77b Donald Byrd, "Graphic Requirements for Music Notation" (unpublished paper, 1977).

BYRD80 Donald Byrd, "Human Engineering in a Machine-Independent Music Notation System", *Proceedings of the 1980 International Computer Music Conference* (Queens College, 1980).

CALC67 "Musical Plotter Knows the Score at University of Toronto", *CalComp Newsletter*, May/June 1967.

CAE83 "New Products" in *Computers and Electronics* (January 1983), p. 11.

CANT71 Donald Cantor, "A Computer Program that Accepts Common Musical Notation", *Computers and the Humanities* 6 (November 1971), pp. 103 – 10.

CARL78 Dan Carlinsky, compiler, *Typewriter Art* (Price/Stern/Sloan, 1978).

CHAF82 Chris Chafe, Bernard Mont-Reynaud, and Loren Rush, "Toward an Intelligent Editor of Digital Audio: Recognition of Musical Constructs", *Computer Music Journal* 6,1 (Spring 1982).

CMJ79 "Products of Interest", *Computer Music Journal* 3,1 (March 1979).

CMJ82 "Products of Interest", *Computer Music Journal* 6,2 (Summer 1982).

DALM75 Armando Dal Molin, "The X-Y Typewriters and Their Application as Music Input Terminals for the Computer", *Proceedings of the Second Annual Music Computation Conference* (Univ. of Illinois, 1975).

DALM78 Armando Dal Molin, "A Terminal for Music Manuscript Input", *Computers and the Humanities* 12 (1978), pp. 287 – 89.

DALM82 Armando Dal Molin, personal communication, July 1982.

DART63 Thurston Dart, *The Interpretation of Music* (Harper and Row, 1963).

DEVA83 Devarahi, "Analog and Digital Sequencers", *Keyboard* 9,4 (April 1983), pp. 26 – 30.

DONA63 Anthony Donato, *Preparing Music Manuscript* (Prentice-Hall, 1963).

DRAK77 G.W.F. Drake and M. Schlesinger, "Vector-Coupling Approach to Orbital and Spin-dependent Tableau Matrix Elements in the Theory of Complex Spectra", *Physical Review A* (May 1977), p. 1993.

DYER80 Charles R. Dyer, Azriel Rosenfeld, and Hanan Samet, "Region Representation: Boundary Codes from Quadtrees", *Communications of the ACM* 23,3 (March 1980), pp. 171 – 79.

ERIC75 Raymond F. Erickson, " 'The DARMS Project': A Status Report", *Computers and the Humanities* 9 (1975), pp. 291 – 98.

FOLE82 James D. Foley and Andries van Dam, *Fundamentals of Interactive Computer Graphics* (Addison-Wesley, 1982)

FOST82 Scott Foster, W. Andrew Schloss, and A. Joseph Rockmore, "Toward an Intelligent Editor of Digital Audio: Signal Processing Methods", *Computer Music Journal* 6,1 (Spring 1982).

FREF83 Freff, "Making Music with the Well-Synthesized PC", *PC Magazine* (December 1983).

FURU82 Richard Furuta, Jeffrey Scofield, and Alan Shaw, "Document Formatting Systems: Survey, Concepts, and Issues", *Computing Surveys* 14,3 (September 1982), pp. 417 – 72.

GILB82 John V. Gilbert, "The Well Tempered McLeyvier: Music Marries the New Technology", *Symphony Magazine* (June/July 1982).

GIPS75 James Gips, *Shape Grammars and Their Uses* (Birkhauser Verlag, 1975).

GOMB75 David Gomberg, *A Computer-Oriented System for Music Printing* (Sc.D. dissertation, Washington University, 1975).

GOMB77 David Gomberg, "A Computer-Oriented System for Music Printing", *Computers and the Humanities* 11 (1977), pp. 63 – 80.

GOOD76 Nelson Goodman, *Languages of Art* (Hackett, 1976).

GOUL70 Murray J. Gould and George W. Logemann, "ALMA: Alphameric Language for Music Analysis" (in [BROO70]).

GREY75 John M. Grey, *An Exploration of Musical Timbre* (Ph.D. dissertation, Stanford University, 1975).

GROV80 Stanley Sadie, ed., *The New Grove Dictionary of Music and Musicians* (20 volumes; Macmillan, 1980).

GSPC79 ACM Graphics Standards Planning Committee, *Status Report of the Graphics Standards Planning Committee*, published as *Computer Graphics* 13,3 (1979).

HASS83 Jeffrey Hass, personal communication, January 1983.

HILL59 Lejaren A. Hiller, Jr., and Leonard Isaacson, *Experimental Music* (McGraw-Hill, 1959).

HILL65 Lejaren A. Hiller, Jr., and R. A. Baker, "Automated Music Printing", *Journal of Music Theory* 9 (Spring 1965), pp. 129 – 50.

HIRS82 Stephen A. Hirsch, "An Algorithm for Automatic Name Placement Around Point Data", *The American Cartographer* 9,1 (1982), pp. 5 – 17.

HOFS79 Douglas R. Hofstadter, *Gödel, Escher, Bach: an Eternal Golden Braid* (Basic Books, 1979).

HOFS82a Douglas R. Hofstadter, "Metamagical Themas: Pattern, Poetry, and Power in the Music of Frédéric Chopin", *Scientific American* 246,4 (April 1982), pp. 16 – 28.

HOFS82b Douglas R. Hofstadter, "Metafont, Metamathematics, and Metaphysics: Comments on Donald Knuth's 'The Concept of a Meta-Font' ", *Visible Language* 16,4 (Autumn 1982), pp. 309 – 38.

HOFS83a Douglas R. Hofstadter, personal communication, April 1983.

HOFS83b Douglas R. Hofstadter, "A Short Statement about Slippability" (unpublished paper, 1983).

HOFS83c Douglas R. Hofstadter, "The Architecture of Jumbo", *Proceedings of the 2nd International Machine Learning Workshop* (Monticello, Illinois, 1983).

HUNT78 Gregory Michael Hunter, *Efficient Computation and Data Structures for Graphics* (Ph.D. dissertation, Princeton University, 1978).

ISO81 International Standards Organization, *Graphical Kernel System (GKS), Version 6.6* (May 1981).

JOHN81 Steven D. Johnson, personal communication, September 1981.

JONE82 Cameron Jones, personal communication, December 1982.

KASS72 Michael Kassler, "Optical Character Recognition of Printed Music: A Review of Two Dissertations", *Perspectives of New Music* (Fall – Winter, 1972).

KERN75 Brian W. Kernighan and Lorinda L. Cherry, "A System for Typesetting Mathematics", *Communications of the ACM* (March 1975), pp. 151 – 56.

KERN78a Brian W. Kernighan and Lorinda L. Cherry, "Typesetting Mathematics — User's Guide", 2nd ed. (Bell Laboratories, Murray Hill, N.J., 1978).

KERN78b Brian W. Kernighan, M. E. Lesk, and J. F. Ossanna, Jr., "Document Preparation", *Bell System Technical Journal* 57,6,part 2 (July – August 1978), pp. 2115 – 36.

KIM81 Scott Kim, *Inversions* (BYTE Books, 1981).

KIRK83 S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing", *Science* 220, pp. 671 – 80 (May 1983).

KNOW71 Prentiss Knowlton, *Interactive Communication and Display of Keyboard Music* (Ph.D. dissertation, Univ. of Utah, 1971).

KNOW72 Prentiss Knowlton, "Capture and Display of Keyboard Music", *Datamation* (May 1972).

KNUT79 Donald E. Knuth, *TEX and METAFONT: New Directions in Typesetting* (Digital Press, 1979).

KNUT81 Donald E. Knuth and Michael F. Plass, "Breaking Paragraphs into Lines", *Software: Practice and Experience* 11 (1981), pp. 1119 – 84.

KNUT82 Donald E. Knuth, "The Concept of a Meta-Font", *Visible Language* 17 (Winter 1982), pp. 3 – 27.

KORN80 William Kornfeld, screen displays of music editor, on covers of *Computer Music Journal* 4,2 and 4,3 (1980).

KORN82 William Kornfeld, personal communication, December 1982.

LEVI84 David Levitt, "Machine Tongues X: Constraint Languages", *Computer Music Journal* 8,1 (1984).

LINC70 Harry Lincoln, ed., *The Computer and Music* (Cornell, 1970).

LONG76 H. C. Longuet-Higgins, "Perception of Melodies", *Nature* 263 (Oct. 21, 1976), pp. 646 – 53.

MAXW82 John Maxwell, *Mockingbird Manual* (Xerox Palo Alto Research Center, 1982).

MAXW83 John Turner Maxwell III and Severo M. Ornstein, *Mockingbird: A Composer's Amanuensis* (Xerox Palo Alto Research Center, 1983).

MAXW84 John Turner Maxwell III and Severo M. Ornstein, "Mockingbird: A Composer's Amanuensis", *Byte* 9,1 (1984).

MEAD80 Carver Mead and Lynn Conway, *Introduction to VLSI Systems* (Addison-Wesley, 1980).

MERC81 Rebecca Mercuri, "MANUSCRIPT: Music Notation for the Apple II", *Proceedings of the 1981 Symposium on Small Computers in the Arts.*

MERR34 *Webster's New International Dictionary,* 2nd ed. (Merriam-Webster, 1934).

MEYR82 Norman Meyrowitz and Andries Van Dam, "Interactive Editing Systems" (two parts), *Computing Surveys* 14,3 (September 1982).

MONT83 Fanya Montalvo, personal communication, May 1983.

MOOR75 James A. Moorer, *On the Segmentation and Analysis of Continuous Musical Sound by Digital Computer* (Ph.D. dissertation, Stanford University, 1975).

MOOR77 James A. Moorer, "On the Transcription of Musical Sound by Computer", *Computer Music Journal* 1,4 (1977).

MOOR82 James A. Moorer, personal communication, December 1982.

NEDC82 New England Digital Corporation, "Synclavier II/Music Printing Option" (1982).

NEDC83a New England Digital Corporation, Synclavier II advertising brochures.

NEDC83b New England Digital Corporation, "Music Printing Option User's Guide", Release B (February 1983).

NELS77 Gary Nelson, "MPL: A Program Library for Musical Data Processing", *Creative Computing* 3,2 (March – April 1977).

NEWM79 William Newman and Robert Sproull, *Principles of Interactive Computer Graphics*, 2nd ed. (McGraw-Hill, 1979).

ORNS82 Severo Ornstein, personal communication, July 1982.

OSSA76 Joseph F. Ossanna, *NROFF/TROFF User's Manual* (Bell Laboratories, Murray Hill, N.J., 1976).

PASS84 Passport Designs, "Polywriter" advertising brochures (1984).

PISZ77 Martin Piszczalski and Bernard A. Galler, "Automatic Music Transcription", *Computer Music Journal* 1,4 (1977).

PISZ81 Martin Piszczalski et al, "Performed Music: Analysis, Synthesis, and Display by Computer", *Journal of the Audio Engineering Society* 29,1/2, pp. 38 – 46.

PRER71 David Prerau, "Computer Pattern Recognition of Printed Music", *Proceedings of the Fall Joint Computer Conference*, 1971.

PRUS66 Dennis Pruslin, *Automatic Recognition of Sheet Music* (Sc.D. dissertation, MIT, 1966).

RASK80 Jef Raskin, "Using the Computer as a Musician's Amanuensis", *BYTE*, vol. 5, nos. 4 – 5 (April – May 1980), pp. 18 – 28 and 120 – 28.

RAST82 Richard Rastall, *The Notation of Western Music* (St. Martin's Press, 1982).

READ69 Gardner Read, *Music Notation*, 2nd ed. (Crescendo, 1969).

READ78 Gardner Read, *Modern Rhythmic Notation* (Indiana, 1978).

REEV78 William Reeves et al, "Ludwig: an Example of Interactive Computer Graphics in a Score Editor", *Proceedings of the 1978 International Computer Music Conference* (Northwestern University), vol. 2, pp. 392 – 409.

REND81 Charles Render, *The Development of a Computer Program to Arrange and Print Traditional Music Notation* (Ed.D. dissertation, Univ. of Illinois, 1981).

ROAD81 C. Roads, "A Note on Music Printing by Computer", *Computer Music Journal* 5,3 (Fall 1981).

ROSS70 Ted Ross, *The Art of Music Engraving and Processing,* 2nd ed. (Hansen, 1970).

SAME80 Hanan Samet, "Region Representation: Quadtrees from Boundary Codes", *Communications of the ACM* 23,3 (March 1980), pp. 163 – 70.

SAME81 Hanan Samet, "An Algorithm for Converting Rasters to Quadtrees", *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3,1 (January 1981), pp. 93 – 95.

SCAL84 Carla Scaletti, personal communication, May 1984.

SEEG58 C. Seeger, "Prescriptive and Descriptive Music Writing", *Musical Quarterly* 44,2 (1958).

SEEG73 Peter Seeger, *Henscratches and Flyspecks* (Berkley, 1973).

SEYB81 Jonathan Seybold, "The Xerox Star: a Professional Workstation", *The Seybold Report on Office Systems* 4,5 (May 1981).

SEYB83 Jonathan Seybold et al, "Digital Typesetter Comparisons", *The Seybold Report on Publishing Systems* 12,9 (17 January 1983).

SHAN57 Howard Shanet, *Learn to Read Music* (Faber and Faber, 1957).

SMIT73 Leland Smith, "Editing and Printing Music by Computer", *Journal of Music Theory* 17 (1973), pp. 292 – 309.

SMIT78 Leland Smith, (MSS User's Manual), (unpublished), 16 April 1978.

SMIT79 Leland Smith, *Handbook of Harmonic Analysis* (San Andreas Press, Palo Alto, 1979).

SMIT82 Leland Smith, personal communication, July 1982.

SPIE83a Laurie Spiegel, personal communication, May 1983.

SPIE83b Laurie Spiegel, letter to the editor, *Computer Music Journal* 7,2 (Summer 1983).

STON80 Kurt Stone, *Music Notation in the Twentieth Century* (Norton, 1980).

SUTH74 I.E. Sutherland, R.F. Sproull, and R.A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms", *Computing Surveys* 6,1 (March 1974).

TRIM81 Stephen Trimberger, "Combining Graphics and a Layout Language in a Single Interactive System", *Proceedings of the 18th Design Automation Conference* (1981).

VICK83 Rochelle Vickey, personal communication, 12 January 1983.

WALL78 Dean Wallraff, "NEDIT: a Graphical Editor for Musical Scores", *Proceedings of the 1978 International Computer Music Conference* (Northwestern University), vol. 2, pp. 392 – 400.

WARN69 John Warnock, "A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures", TR 4-15 (Univ. of Utah Computer Science Dept., 1969).

WATK82 William A. Watkins, personal communication, 8 September 1982.

WEBB83 Robert Webber, personal communication, March 1983.

WENK70 Jerome Wenker, "A Computer Oriented Music Notation including Ethnomusicological Symbols" (in [BROO70]).

WENK74 Jerome Wenker, "MUSTRAN II: A Foundation for Computational Musicology", *Computers in the Humanities*, L. Mitchell, ed. (Edinburgh, 1974).

WINO79 Allen Winold and John Rehm, *Introduction to Music Theory*, 2nd ed. (Prentice-Hall, 1979).

WITT74 Gary Wittlich et al, "Non-Physics Measurements on the PEPR System: Seismograms and Music Scores", *Report to the Oxford Conference on Computer Scanning* (Oxford: Nuclear Physics Laboratory, 1974), pp. 487 – 489.

WITT78 Gary Wittlich, Donald Byrd, and Rosalee Nerheim, "A System for Interactive Encoding of Music Scores Under Computer Control", *Computers and the Humanities* 12 (1978), pp. 309 – 319.

WITT82 Gary Wittlich, personal communication, December 1982.

WORL64 *Webster's New World Dictionary* (World, 1964).

XENA71 Iannis Xenakis, *Formalized Music* (Indiana University Press, 1971).

Appendix I

```
          =
        IUCMS
        ==  =
        =    =
        =    =
        =   =
        =  ==
        = ==
        ===
        ==
       ==
      == =
    ===  =
   ===  MUSIC==
  ==   IUCMSIUCMS
 ===  == =     ===
 ==  ==  =      ===
=== ==   =       ==
 == ==   =       ==
 ==   =  =      ==
  ==   = =     ==
    MUSICMUSIC==
      IUCMS=
         =
         =
     ==  =
    ====  =
    MUSIC=
     ====
```

SMUT 2.9

USER'S GUIDE

DONALD BYRD
31 JANUARY 1983

INDIANA UNIVERSITY
COMPUTER MUSIC SYSTEM
DOCUMENT NO. 2

SMUT USER'S GUIDE

INTRODUCTION

THE FOLLOWING MATERIAL IS EXTRACTED FROM THE SMUT MAIN PROGRAM.

```
      PROGRAM SMUT(INPUT=200, OUTPUT=200, PLOT=200, TAPE9=200,
     $ ZSMUT1=520, ZSMUT2=520, ZSMUT3=520, ZSMUTX=520, TAPE5=INPUT,
     $ TAPE6=OUTPUT, TAPE8=PLOT, TAPE10=ZSMUT1, TAPE11=ZSMUT2,
     $ TAPE12=ZSMUT3, TAPE13=ZSMUTX)
C
C
C
C
C                         TO IANNIS XENAKIS
C
C                              S M U T
C                          SYSTEM FOR MUSIC
C                           TRANSCRIPTION
C                           -------------
C                            DONALD BYRD
C                         INDIANA UNIVERSITY
C
C  BEGUN SEPTEMBER 1968...SMUT VERSION 1.2, JULY 1975
C  POLYPHONIC VERSION 2.0, SEPTEMBER 1977.
C  VERSION 2.8 TO SUPPORT SHARED STAVES, MARCH 1982.
C  VERSION 2.9, JANUARY 1983.
C
CCCCCCCCCCCCCC
C            C
C   XXXXXXX  C
C  X       X C
C  X         C
C  X         C
C  X         C
C  X         C    COPYRIGHT (C) 1972,1975,1976,1977,1978,1979,
C  X         C    1980,1981,1982, DONALD BYRD
C  X         C
C  X         C
C  X       X C
C   XXXXXXX  C
C            C
CCCCCCCCCCCCCC
C
C
C  COMMENTS, QUESTIONS, ETC. ARE WELCOME AND SHOULD BE ADDRESSED
C  TO THE AUTHOR AT
C     KURZWEIL MUSIC SYSTEMS
C     411 WAVERLEY OAKS ROAD
C     WALTHAM, MASSACHUSETTS  02154
C  OR CALL 617-893-5900.
C
C  THIS PROGRAM TRANSLATES ALPHANUMERIC DATA INTO STANDARD MUSICAL
C    NOTATION AND OUTPUTS IT TO A DIGITAL PLOTTER OR SIMILAR DE-
```

```
C    VICE.  SMUT 2.9 CAN DRAW SCORES WITH ONE OR TWO VOICES PER
C    STAFF, OR PARTS WITH ONE VOICE.  IN SCORES, ALL BARLINES MUST
C    COINCIDE.
C  A DESCRIPTION OF THE DATA FORMATS MAY BE FOUND IN THE LISTINGS
C    OF SUBROUTINES  INITSM  AND  PASS1 .  NOTE, HOWEVER, THAT THE
C    DATA FORMAT IS RATHER CLUMSY TO PREPARE MANUALLY AND THERE
C    WILL RARELY BE ANY NEED TO DO SO, SINCE PROGRAMS EXIST (SMIRK
C    AND JANUS) THAT TRANSLATE MUSTRAN DATA AND MUSIC V NOTE CARDS
C    INTO SMUT COMMANDS.
C  FOR FILE USAGE AND INSTALLATION PARAMETERS, SEE THE BLOCK DATA
C    MODULE.
C  IMPORTANT GENERAL COMMENTS ON THE STRUCTURE OF THE PROGRAM ARE
C    IN THIS ROUTINE,  WRITE1 , PASS2 , PASS3 , AND  JUSTIF .
C  SMUT  AND RELATED PROGRAMS (INCLUDING MUSTRAN, MUSIC V, SMIRK,
C    AND JANUS) ARE DISCUSSED IN DONALD BYRD, 'AN INTEGRATED COM-
C    PUTER MUSIC SOFTWARE SYSTEM', COMPUTER MUSIC JOURNAL 1,2 (JULY
C    1977).  A STANDARD MUSIC NOTATION TEXT WHICH I HAVE OFTEN IG-
C    NORED HEREIN IS GARDNER READ, MUSIC NOTATION, 2ND EDITION
C    (BOSTON--CRESCENDO, 1969).  SEE ALSO TED ROSS, THE ART OF
C    MUSIC ENGRAVING AND PROCESSING (MIAMI--HANSEN, 1970).
C
C  'THIS PROGRAM I HAVE LEARNED FROM MY USERS.'  I AM GRATEFUL TO
C    ALL MY COLLEAGUES AT THE WRUBEL COMPUTING CENTER, INDIANA
C    UNIVERSITY, AS WELL AS TO THE MUSICIANS WHO HAVE STRUGGLED
C    WITH  SMUT .  I ESPECIALLY WISH TO THANK JEFF HASS, JIM HETT-
C    MER, DOUG HOFSTADTER, DAVE KRIEWALL, ROSALEE NERHEIM, BRUCE
C    ROGERS, DAVID ROTH, AND GARY WITTLICH FOR MANY HELPFUL COM-
C    MENTS AND FOR GENERAL MORAL SUPPORT.
C
C
C  THE FOLLOWING COMMENTS ARE FOR THE BENEFIT OF THOSE FAMILIAR
C    WITH EARLIER VERSIONS OF SMUT.
C
C  DIFFERENCES BETWEEN SMUT 2.9 AND SMUT 2.0 (YOU DECIDE WHETHER
C    THEY'RE INCOMPATIBILITIES OR NOT)...
C    1. THE R COMMAND NOW HAS AN OPTION FOR DESCRIBING MULTI-BAR
C       RESTS, AND ITS Y-POSITION PARAMETER IS NOW IN HALF-SPACES
C       (NOT SPACES).
C    2. A NEW OPTION, KIND=+2, HAS BEEN ADDED TO THE 'S' COMMAND.
C    3. A Y-POSITION PARAMETER LIKE THAT FOR THE 'A', 'P', AND 'R'
C       COMMANDS HAS BEEN ADDED TO THE 'T' COMMAND.
C    4. SOME OPTIONS HAVE BEEN ADDED TO THE 'B' COMMAND, AND THE
C       MEANING OF THE PARAMETER'S SIGN (WITH AUTOBARRING) RE-
C       VERSED.
C    5. 12 HEADER/FOOTER LINES ARE ALLOWED INSTEAD OF 5.
C    6. ADDITIONAL LAYOUT LINE PARAMETERS INDICATE WHAT QUALITY
C       IS NEEDED AND CHARACTERISTICS OF THE PLOTTER BEING USED.
C    7. SOME NOTE-RELATED SYMBOLS ARE NOW ALWAYS PLACED ABOVE THE
C       STAFF (E.G. UPBOW, DOWNBOW) WHILE OTHERS GO ABOVE OR BE-
C       LOW THE NOTEHEAD (E.G. ARTICULATION MARKS).
C    8. THE FIRST COMPLETE BAR IS NOW LABELLED BAR 1, RATHER THAN
C       A PARTIAL BAR (WHEN THERE IS AN ANACRUSIS).
C    9. IF A SYSTEM IS NOT RIGHT-JUSTIFIED, ONLY THE PORTION OF
C       THE STAVES ACTUALLY USED IS DRAWN.
C   10. THREE NEW CONTROL COMMANDS HAVE BEEN ADDED - '* HCROWD'
C       TO CONTROL HORIZONTAL SPACING, AND '* MEASNO'/'* NOMEASNO'
```

```
C         TO DETERMINE WHETHER MEASURE NUMBERS SHOULD BE WRITTEN ON
C         THE MUSIC OR NOT.
C     11. A PARAMETER HAS BEEN ADDED TO THE 'S' COMMAND THAT MAKES
C         SOME TIME SIGNATURES NOT CONSISTING OF NUMBER-ABOVE-NUMBER
C         (E.G.  'C', THE SINGLE NUMBER '3') DOABLE.
C     12. RHYTHM DECOMPOSITION HAS BEEN GENERALIZED TO HANDLE ANY
C         METER WITH NO MORE THAN 10 BEATS PER MEASURE.
C     13. CONTROL COMMANDS '* SHARP' AND '* RHLIN' HAVE BEEN CHANGED
C         TO '* NOFLAT' AND '* NORHDEC', RESPECTIVELY.
C     14. THE DEFAULTS FOR LAYOUT LINE PARAMETERS  HS , YDELT  HAVE
C         BEEN CHANGED.
C     15. THREE NOTE-ASSOCIATED SYMBOLS ('AH','TW','MR') HAVE BEEN
C         ADDED.
C     16. AN INITIALIZATION LINE CAN NOW SPECIFY HOW VOICES ARE
C         GROUPED, AND THERE IS AN OPTION ON P COMMANDS TO MAKE
C         THEIR EXECUTION CONDITIONAL ON VOICE GROUPING.  VOICE
C         GROUPING INCLUDES HAVING TWO VOICES SHARE A STAFF.
C     17. THE LAYOUT LINE MUST HAVE AN 'L' IN COLUMN 1.
C     18. RESTS CAN NOW HAVE FERMATI AND PAUSES AS ASSOCIATED SYM-
C         BOLS.
C     19. A VERSION OF  READ4 , THE PASS4 INPUT SUBROUTINE, CONTAIN-
C         ING A BUILT-IN EDITOR IS NOW AVAILABLE.  THIS MAKES IT
C         POSSIBLE TO DO A GREAT DEAL THAT COULD NOT BE DONE OTHER-
C         WISE.
C     20. THE LEGAL PITCH RANGE HAS BEEN REDUCED TO A MORE REASON-
C         ABLE ONE.
C     21. THE P COMMAND NOW ALLOWS A CHOICE OF FONTS (E.G., FOR DY-
C         NAMICS).
C     22. A RUNNING FOOTER COMMAND NOW ALLOWS SPECIFYING A LINE TO
C         BE WRITTEN ON EVERY PAGE OF MUSIC (AFTER THE FIRST).
C     23. PERFORMANCE DIRECTIONS CAN GO UP TO SIZE 5 INSTEAD OF 4.
C     24. CONTROL OF BEAM EXTENT IS NOW POSSIBLE WITH '* BEAM' .
C     25. THERE ARE NEW CONTROL COMMANDS '* SET', '* IFEQ', AND
C         '* ENDIF' FOR CONDITIONAL EXECUTION.
C     26. CONTROL OF SLUR CURVATURE AND DIRECTION IS NOW POSSIBLE.
C     27. SQUARE, DIAMOND, 'X' SHAPED AND OMITTED NOTEHEADS ARE NOW
C         AVAILABLE.
C     28. AN 'M' (MISCELLANEOUS SYMBOL) COMMAND CAN SPECIFY CERTAIN
C         SYMBOLS INDEPENDENT OF A NOTE OR REST, OR LEAVE HORIZONTAL
C         SPACE.
C     29. AN 'E' COMMAND CAN BE USED TO GIVE COMMANDS TO THE EDITING
C         VERSION OF  READ4  DIRECTLY FROM THE REGULAR SMUT INPUT
C         FILE.
C     30. A NEW INITIALIZATION LINE CAN SPECIFY HOW MANY BARS TO PUT
C         IN EACH SYSTEM OF MUSIC.
C     31. THE 'A' (ARTIFICIAL GROUP) COMMAND HAS A NEW OPTION TO
C         SUPPRESS PRINTING OF ITS ACCESSORY NUMERAL.
C     32. LAYOUT LINE OPTIONS HAVE BEEN ADDED TO RUN ONLY PART OF
C         THE PROGRAM AT A TIME.  THIS IS IN SUPPORT OF EXTERNAL
C         EDITING.
C     33. THERE ARE NEW CONTROL COMMANDS TO SET TIME USED OR REMAIN-
C         ING IN A MEASURE FOR USE WITH ANACRUSES, NONSTANDARD NOTA-
C         TION, ETC.  THIS FUNCTION FORMERLY WAS PERFORMED BY THE
C         TNOW  PARAMETER ON THE LAYOUT LINE, WHICH IS NOW GONE.
C     34. INTEGERS ARE NOW READ WITH TRAILING BLANKS IGNORED INSTEAD
```

```
C        OF BEING TREATED AS ZEROS.
C    35. THERE ARE NEW CONTROL COMMANDS TO TURN JUSTIFICATION ON
C        AND OFF AND TO GIVE BETTER CONTROL OF HORIZONTAL SPACING.
C    36. THE REST COMMAND HAS AN 'INVISIBILITY' OPTION.
C    37. THE 'S' COMMAND NOW HAS A PARAMETER TO CONTROL VERTICAL
C        SPACING AND A 'START NEW PAGE' OPTION.
C    38. SEVERAL PARAMETERS HAVE BEEN ADDED TO THE 'I' COMMAND SO
C        IT NOW CONTROLS STAFF INDENTING AS WELL AS IDENTIFI-
C        CATION, AND CONTROL OF THE SIZE OF AND FONT OF THE IDENT.
C        IS NOW POSSIBLE.
C    39. A NEW '* REPKS' CONTROL COMMAND AFFECTS REPETITION OF THE
C        OLD KEY SIGNATURE AT THE BEGINNING OF THE STAFF WHEN THE
C        KEY SIGNATURE CHANGES IN THE FIRST MEASURE OF THE STAFF.
C    40. THE  STAFF  AND  NOSTAFF  CONTROL COMMANDS HAVE BEEN RE-
C        NAMED  SYST  AND  NOSYST .
C    41. A '* SHARE' CONTROL COMMAND HAS BEEN ADDED FOR SPECIAL
C        EFFECTS.
C    42. EXPLICIT BEGINNING AND END POINTS FOR BEAMS CAN BE SPEC-
C        IFIED.
C    43. THERE IS NOW A PARAMETER TO SET THE BOTTOM MARGIN ON THE
C        LAYOUT LINE.
C   MANY MINOR IMPROVEMENTS HAVE ALSO BEEN MADE IN THE LISTING,
C     ERROR CHECKING, GRAPHIC QUALITY OF THE NOTATION, ETC.
C
```

LAYOUT AND INITIALIZATION LINES

DATA FOR A SMUT RUN CONSISTS OF EXACTLY ONE LAYOUT LINE, A SMALL NUMBER OF INITIALIZATION LINES, AND ANY NUMBER OF MUSIC DATA LINES, IN THAT ORDER.

A. THE LAYOUT LINE DESCRIBES THE PAGE FORMAT.
(ALL DISTANCES, LENGTHS, ETC. ARE GIVEN IN INCHES)

COL.	FORMAT	NAME	INFORMATION	NORMAL DEFAULT
1	A1	IEORL	'L'=NORMAL, 'E'=EDIT MODE (SEE BELOW)	
2	A1	IFSCOR	IF EDIT MODE, 'F'=RUN PASS IV ONLY (REQUIRES AUXILIARY FILES). IN BOTH MODES, 'P'=DO A SET OF PARTS, 'S'=DO A SCORE. 'P' MAY ALSO BE USED FOR A GROUP OF UNRELATED 1-VOICE EXAMPLES.	S
3-6	I4	LISTC	LISTING CONTROL (NEGATIVE=SHORT LISTING. . .0=NORMAL, PRINTS BACK INPUT. . . POSITIVE=NORMAL PLUS DEBUG PRINT).	(NONE)
7-11	F5.2	RESOL	RESOLUTION OF THE PLOTTER USED (SEE BELOW)	.005
12-16	F5.2	HS	HEIGHT OF EACH STAFF	.24
17-21	F5.2	XLN	LENGTH OF THE STAFF	7.70
22-26	F5.2	YH	Y-COORDINATE OF THE BOTTOM LINE OF THE TOP STAFF ON EACH PAGE AFTER 1ST	9.60
27-31	F5.2	YDELT	VERTICAL DISTANCE BETWEEN STAVES, BOTTOM TO BOTTOM	.96
32-36	F5.2	YMIN	THE LOWEST POSSIBLE POSIION FOR THE BOTTOM OF A STAFF. VERY ROUGHLY SPEAKING, THIS IS THE BOTTOM MARGIN.	.4
37-41	I5	MAXERR	MAXIMUM NO. OF NONFATAL ERRORS	25
42	A1	MODPUB	BLANK=NORMAL, 'P'=PUBLICATION QUALITY DESIRED. SEE BELOW.	(NONE)
43-44	I2	LINWID	USED WITH RESOL TO DETERMINE LINEWIDTH. SEE BELOW.	10

INTERNALLY, SMUT CONSISTS OF FOUR 'PASSES', THE LAST OF WHICH ACTUALLY DRAWS THE MUSIC (SEE 'PROGRAM OPERATION' BELOW FOR MORE DETAILS). IN NORMAL MODE, ALL FOUR PASSES ARE RUN. IN EDIT MODE, IF IFSCOR='S' OR 'P' ONLY THE FIRST THREE ARE RUN AND TWO INTERMEDIATE FILES DESCRIBING THE SCORE OR SET OF PARTS ARE GENERATED. IN EDIT MODE, IF IFSCOR='F', ONLY PASS IV IS RUN, AND THE TWO INTERMEDIATE FILES (PRESUMABLY GENERATED BY A PREVIOUS RUN) MUST BE AVAILABLE. AGAIN, SEE 'PROGRAM OPERATION' FOR MORE DETAILS.

THE TABLE ABOVE HAS THE HEADING 'NORMAL' DEFAULT BECAUSE, WHEN PASS IV ONLY IS RUN, DEFAULTS ARE TAKEN FROM THE LAYOUT LINE USED WHEN PASSES I THRU III WERE RUN. IN ANY CASE, THE DEFAULT VALUE, WHERE THERE IS ONE, WILL BE SUBSTITUTED FOR ANY PARAMETER THAT IS BLANK. IN ADDITION, VALUES ARE RESTRICTED AS INDICATED IN THE DESCRIPTION OF ERROR MESSAGE F02 .

RESOL AND LINWID TELL SMUT ABOUT THE PLOTTER TO BE USED SO IT CAN PRODUCE THE BEST QUALITY PLOT WITHOUT WASTING COMPUTER OR PLOTTER TIME. SPECIFICALLY, RESOL IS THE RESOLUTION OF THE PLOTTER, E.G. .005 FOR VERSATEC 1200, .002 FOR CALCOMP 1037, ETC. ORDINARILY, THE WIDTH OF A LINE ON A PLOTTER IS THE SAME AS ITS RESOLUTION, BUT ON A MECHANICAL (PEN-AND-INK) PLOTTER SUCH AS THE CALCOMP 1037, MOST PENS ARE WIDER THAN THIS. SMUT USES THE LINEWIDTH AS THE SPACING BETWEEN THE PARALLEL LINES IT DRAWS TO BLACKEN MANY SYMBOLS IN. SO LINWID CAN BE USED TO SPECIFY A LINEWIDTH LARGER THAN THE RESOLUTION VIA THE FORMULA

LINEWIDTH = RESOL*.1*ABS(LINWID) .

FINALLY, IF MODPUB IS 'P', SMUT TAKES EXTRA CARE IN DRAWING SOLID NOTEHEADS AND IN THICKENING EVERYTHING (ESPECIALLY SLURS, CLEFS, AND QUARTER RESTS) IN AN EFFORT TO PRODUCE THE BEST POSSIBLE PLOT QUALITY. THIS SLOWS THE PROGRAM DOWN AND IS LIKELY TO HELP ONLY ON HIGH-RESOLUTION VECTOR-DRAWING PLOTTERS, ESPECIALLY PEN-AND-INK PLOTTERS.

SMUT INSISTS ON HS BEING AT LEAST 15 TIMES AS LARGE AS RESOL . ACTUALLY, FOR READABLE RESULTS, IT SHOULD BE AT LEAST 20 OR, BETTER, 30 TIMES AS LARGE.

SMUT RELIES ON YMIN TO PREVENT THE PLOT FROM GOING OFF THE BOTTOM OF THE PAGE. THIS IS BY NO MEANS FOOLPROOF, ESPECIALLY IF THE MUSIC HAS VERY LOW NOTES OR PERFORMANCE DIRECTIONS WELL BELOW THE STAFF. EXPERIMENTATION WITH YMIN MAY BE NECESSARY FOR BEST RESULTS.

SPACING BETWEEN SYSTEMS IN A SCORE IS AFFECTED BOTH BY YDELT AND BY THE NUMBER OF STAVES. HOWEVER, IT WILL ALWAYS BE SOMEWHERE BETWEEN (NO. OF STAVES)*YDELT AND (NO. OF STAVES+1)*YDELT --I.E., UP TO YDELT MORE SPACE IS LEFT BETWEEN SYSTEMS THAN BETWEEN STAVES WITHIN A SYSTEM. SPECIFICS APPEAR IN THE FOLLOWING TABLE.

NO. OF STAVES	SPACE BETWEEN SYSTEMS
1	1*YDELT
2	2.25*YDELT
3	3.5*YDELT
4	4.75*YDELT
5 OR MORE	(NO. OF STAVES+1)*YDELT

IT IS POSSIBLE, ALTHOUGH SOMEWHAT MESSY, TO COMPUTE FROM THIS HOW MANY SYSTEMS SMUT WILL PUT ON A PAGE, GIVEN THE NUMBER OF STAVES IN THE SYSTEM, YDELT , AND YH .

FOLLOWING ARE SEVERAL SAMPLE LAYOUT LINES, WITH NAMES OF OUTPUT DEVICES THEY MIGHT BE SUITABLE FOR.

L.	LIST	RESOL	HS...	XLN..	YH...	YDELT	YMIN.	MAXER	MLW	OUTPUT DEVICE
LS	0	.005	.24	7.70	9.60	.96	.40	20	00	(DEFAULTS)
LP	0	.010	.60	6.60	4.8	1.5		-1	00	DEI+ADM3A/TEK,BIG
LS	0	.010	.40	6.60	4.8	1.2			00	DEI+ADM3A/TEK,MED.
LS	0	.05	1.8	32.0	8.0	4.0		-1	00	LINE PRINTER,BIG
LS	0	.05	1.0	31.5	9.2	2.7		-1	00	LINE PRINTER,MED.
LS	-1	.002							P15	CALCOMP 1037,NARROW
LS	-1	.002	.27	10.2	12.8	.94			P15	CALCOMP 1037,WIDE
LS	-1	.002	.31	11.5	14.4	1.05			P15	CALCOMP 1037,67PCT.
LS	0	.005							00	VERSATEC 1200
LS	0	.005	.20			.80			00	VERSATEC 1200,SMALL

B. INITIALIZATION LINES ARE EITHER 'H' (HEADER), 'F' (FOOTER), 'R' (RUNNING FOOTER), 'G' (GROUP VOICES), 'B' (BARS PER SYSTEM), 'C' (COMMENT), OR 'X' (END INITIALIZATION LINES AND BEGIN MUSIC DATA).

1. HEADER/FOOTER LINES TELL SMUT WHAT, IF ANYTHING, TO WRITE AT THE TOP OF THE FIRST PAGE OF MUSIC (TYPICALLY THE TITLE OF THE PIECE AND THE COMPOSER'S NAME) AND AT THE BOTTOM OF THE FIRST PAGE (TYPICALLY A COPYRIGHT NOTICE AND/OR GENERAL PERFORMANCE INSTRUCTIONS). THE FORMAT OF EACH LINE IS--

COL. INFORMATION

1 'H' (HEADER LINE, INFORMATION TO GO AT TOP OF PAGE) OR 'F' (FOOTER LINE, TO GO AT BOTTOM OF PAGE).

2 A DIGIT FROM 1 THRU 9 GIVING THE SIZE OF THE CHARACTERS TO BE USED FOR THIS LINE, RELATIVE TO THE HEIGHT OF THE STAFF. 1 TO 4 ARE THE SAME AS PERFORMANCE DIRECTION SIZES 1 TO 4, 8 MEANS CHARACTERS THE SAME HEIGHT AS THE STAFF, 9 IS EVEN LARGER. RECOMMENDED SIZES ARE...

	IN SCORE	IN PARTS
TITLE	7 OR 8	6
COMPOSER'S NAME, ETC.	5 OR 6	4

3-80 ON 'H' AND 'F' LINES, SPECIFICATION OF FROM ZERO TO THREE SUBLINES, EACH PRECEDED BY ONE OR MORE BLANKS. THE NOTATION OF A SUBLINE IS BEST DESCRIBED WITH AN EXAMPLE--

```
    R'BENJAMIN BRITTEN'
```

MEANS, WRITE 'BENJAMIN BRITTEN' RIGHT-JUSTIFIED ON THE LINE. BESIDES 'R', POSITION CODES ARE 'L' (LEFT-JUSTIFY) AND 'C' (CENTER). THUS, A COMPLETE LINE MIGHT BE--

```
   H7  L'WILFRED OWEN' R'BENJAMIN BRITTEN'
```

I.E., WRITE AT THE TOP OF THE PAGE IN RATHER LARGE LETTERS A LINE WITH 'WILFRED OWEN' FLUSH AGAINST THE LEFT MARGIN AND 'BENJAMIN BRITTEN' FLUSH AGAINST THE RIGHT. ANY CHARACTER NOT APPEARING IN THE SUBLINE CAN BE USED TO DELIMIT ITS BEGINNING AND END, SO IT IS POSSIBLE TO HAVE TITLES THAT INCLUDE THE ' --FOR EXAMPLE,

```
          C/MILES' MODE/
```

AN '=' WHERE A POSITION CODE IS EXPECTED TELLS SMUT TO IGNORE THE REST OF THE LINE. THIS ALLOWS (E.G.) PUTTING SEQUENCE NUMBERS ON HEADER/FOOTER LINES.

NO MORE THAN 12 HEADER AND FOOTER LINES MAY APPEAR.

2. THE 'R' (RUNNING FOOTER) LINE GIVES INFORMATION TO BE WRITTEN AT THE BOTTOM OF EVERY PAGE AFTER THE FIRST. ITS FORMAT IS IDENTICAL TO THAT OF A HEADER/FOOTER LINE, EXCEPT THAT A RIGHT-JUSTIFIED SUBLINE SHOULD NOT BE USED, SINCE THE PAGE NUMBER IS WRITTEN AT THE RIGHT END OF THE RUNNING FOOTER LINE. SIZE 4 IS RECOMMENDED. ONLY ONE RUNNING FOOTER LINE SHOULD APPEAR.

3. A 'G' LINE TELLS SMUT HOW TO GROUP VOICES IN A SCORE, INCLUDING WHICH VOICES SHARE STAVES. IT CONTAINS A 'G' IN COLUMN 1, AND COLUMNS 2 THRU 80 CONTAIN GROUPING INFORMATION. ONE OR MORE BLANKS INDICATE THE END OF A GROUP. THE SPECIFICATION OF EACH GROUP IS EITHER A SERIES OF '1'S AND '2'S, WHERE '1' MEANS UNSHARED AND '2' MEANS SHARED STAFF, OR IS A DIGIT 3 THRU 9 GIVING THE NUMBER OF UNSHARED STAVES. FOR EXAMPLE

```
     G 4 3
```

INDICATES FOUR VOICES IN THE FIRST GROUP AND THREE IN THE SECOND, EACH WITH ITS OWN STAFF.

```
     G 22
```

INDICATES A SINGLE GROUP OF FOUR VOICES ON TWO SHARED STAVES, E.G. FOR A CHORALE. FINALLY,

```
     G 1222 22 1 5
```

MEANS FOUR GROUPS, RESPECTIVELY WITH SEVEN, FOUR, ONE, AND FIVE VOICES. VOICE 1 HAS ITS OWN STAFF, VOICES 2 AND 3 SHARE A STAFF, VOICES 4 AND 5 SHARE ANOTHER, ETC. - THIS MIGHT BE A MOZART ORCHESTRA. THE SAME THING COULD BE WRITTEN

```
     G 1222   22   1   11111
```

ANY GROUPING SPECIFIED FOR MORE VOICES THAN THERE IS DATA FOR IS IGNORED.

VOICE GROUPING AFFECTS HOW BARLINES ARE DRAWN - THEY ARE INTERRUPTED BETWEEN GROUPS - AND HANDLING OF CONDITIONAL PERFORMANCE DIRECTIONS, WHICH ARE WRITTEN ONLY FOR THE TOP OR ONLY VOICE OF EACH GROUP.

VOICES SHARING A STAFF HAVE THEIR STEMS POINTED OUTWARDS, SLURS AND GROUPET ACCESSORY NUMERALS PUT ON THE OUTSIDE, ETC. THEIR CLEFS, KEY SIGNATURES, AND TIME SIGNATURES MUST ALWAYS AGREE.
N.B. SMUT 2.9 IS TOTALLY IGNORANT OF THE DANGER OF VOICES SHARING A STAFF COLLIDING. IN PARTICULAR, WHEN THEIR NOTES ARE A SECOND APART OR IN UNISON, THE HEADS WILL OVERLAP OR BE SUPERIMPOSED, AND WHEN BOTH HAVE ACCIDENTALS AND ARE WITHIN A FOURTH, THE ACCIDENTALS WILL OVERLAP.
DEFAULT WHEN DOING A SCORE IS THAT ALL VOICES FORM ONE GROUP AND EACH HAS ITS OWN STAFF. 'G' LINES ARE ILLEGAL WHEN DOING PARTS.

4. 'B' (BAR-PER-SYSTEM) LINES CONTAIN IN COLUMNS 2 THRU 80 A SERIES OF UP TO 60 DIGITS SPECIFYING HOW MANY BARS ARE TO BE PUT IN EACH SYSTEM OF MUSIC. BLANKS ARE IGNORED. AN '=' MEANS THE THE END OF THE LIST. OCCURENCE OF A 'B' LINE PERMANENTLY TURNS OFF SMUT'S NORMAL MECHANISM FOR DECIDING HOW MANY BARS TO PUT IN EACH SYSTEM (ALTHOUGH IT WILL STILL TRY TO PREVENT DISASTEROUS OVERCROWDING). FOR EXAMPLE,

```
  B 2222333      -OR-      B 2222 3 3 3
```

SPECIFIES 2 BARS IN EACH OF THE FIRST 4 SYSTEMS AND 3 BARS IN EACH OF THE NEXT 3. IF MORE SYSTEMS ARE REQUIRED, I.E. IF THERE ARE MORE THAN 17 BARS, THE REST OF THE MUSIC WILL BE SQUEEZED UNMERCIFULLY.

5. COMMENT LINES ARE PRINTED OUT BUT IGNORED. THEY CAN CONTAIN ANYTHING IN COLUMNS 2-80.

TO DEMONSTRATE ALL THIS, A LEGITIMATE RUN MIGHT BEGIN LIKE THIS...

```
   LP                                          (LAYOUT LINE)
   H7 C/SYEEDA'S SONG FLUTE/
   H5 R'JOHN COLTRANE'
   X
   S  1  1  0  4  4   =                        (BEGIN MUSIC DATA)
   .
   .
   .
```

OR LIKE THIS...

```
   LS  -1 .002 .20            .75              (LAYOUT LINE)
   H8 C'WAR REQUIEM'                           (BEGIN INIT. LINES)
   H6 L'WILFRED OWEN' R'BENJAMIN BRITTEN, OP. 66'
   H6                                          (TO LEAVE SPACE)
   H7 C'REQUIEM AETURNUM'
   F4 C'COPYRIGHT 1962 BY BOOSEY AND HAWKES'
   R4 L'WAR REQUIEM'
   G 122122 222 3 2 5
   X                                           (END INIT. LINES)
   S  1  1 -1  5  4    =                       (BEGIN MUSIC DATA)
   .
   .
   .
```

MUSIC DATA LINES

N.B. THIS SECTION GIVES THE EXACT FORMAT OF SMUT MUSIC-DESCRIBING COMMANDS. SINCE THESE ARE GENERATED AUTOMATICALLY BY SMIRK AND BY JANUS , THE PROGRAMS THAT INTERFACE MUSTRAN AND COMPOSING PROGRAMS TO SMUT , USERS OF SMIRK OR JANUS NEED NOT PAY TOO MUCH ATTENTION TO THIS SECTION.

```
C  EACH DATA LINE (OR LINE IMAGE) IS DIVIDED INTO FOUR 20-COLUMN
C    DIVISIONS.  A COMMAND MAY INCLUDE 1, 2, OR 3 OF THESE DI-
C    VISIONS.  COLUMN 1 ALWAYS BEGINS A COMMAND.  A '$' OR SEMICOLON
C    IN COLUMN 20 INDICATES A NEW COMMAND BEGINS IN 22, AND SIMILAR-
C    LY FOR COLUMNS 40-41 AND 60-61.  AN '=' IN COLUMN 20, 40, OR
C    60 INDICATES THE REST OF THE LINE IS TO BE IGNORED.
C  THE TYPE OF INFORMATION IN A COMMAND AND HOW IT IS FORMATTED IS
C    DETERMINED BY THE 1ST CHARACTER OR OP CODE (IN COLUMN 1, 21,
C    41, OR 61), AS FOLLOWS -
C
C    OP CODE  COMMAND                          NO. OF DIVS., FORMAT
C
C     N       NOTE                             (1)4I3,3A2 (2)4I3,5A2
C     G       GRACE NOTE                       (1) 4I3
C     R       REST                             (1) 3I3,A2
C     B       BARLINE                          (1) I3
C     P       PERFORMANCE DIRECTION            (1)3I3,2A4,A1 (2)3I3,
C                                                 7A4,A1 (3)3I3,12A4
C     A       ARTIFICIAL GROUP                 (1) 3I3
C     S       STAFF-CLEF-KEY SIGNATURE-METER   (1) 7I3
C     I       IDENTIFICATION                   (1)I3,4A4 (2)I3,8A4
C     *       CONTROL                          (1) I3, 2A4
C     E       EDIT                             ( )I1,I2,A1,I5,2(1X,I2),
C                                                 9A5,3A2
C     M       MISCELLANEOUS SYMBOL             (1) I3,A2
C     U       CALL USER ROUTINE                (1) 2I3,I4,I3
C     T       OCTAVA                           (1) 2I3
C     X       END-OF-VOICE                     (1) A1
C    (BLANK)  NO-OP
C    ANYTHING ELSE IS ILLEGAL.
C
C
C    E X P L A N A T I O N   O F   P A R A M E T E R S
C
C    N.B. GENERALLY A STATEMENT THAT SOMETHING 'CANNOT' BE DONE
C    MEANS IF IT IS, AN ERROR MESSAGE WILL BE ISSUED.  A STATEMENT
C    THAT SOMETHING 'SHOULD NOT' BE DONE USUALLY MEANS THAT IF IT
C    IS, SMUT WILL NOT CATCH IT AND BAD THINGS MAY HAPPEN.
C
C    N.B. THE 'I' FORMAT USED HERE IS LIKE FORTRAN 77, NOT FORTRAN
C    66, IN THAT FORTRAN 66 TREATS TRAILING BLANKS LIKE ZEROS,
C    WHEREAS  SMUT AND FORTRAN 77 IGNORE THEM (WHICH MAKES FAR MORE
C    SENSE).
C
```

```
C  NAME   FORMAT  MEANING
C
C  ** NOTE (N) COMMAND **
C  NOTP     I3    THE MEANING OF  NOTP  AND  JQP  DEPENDS ON WHETHER
C                 CHROM  IS IN EFFECT.  IF SO, NOTP  IS AN  INTEGER,
C                 OPTIONALLY SIGNED, REPRESENTING THE NOTE'S PITCH IN
C                 SEMITONES RELATIVE TO THE PIANO'S LOW A, NOTP=0.
C                 THUS, MIDDLE C IS 39, THE PIANO'S TOP C 87, AND SO
C                 ON.  LEGAL RANGE IS -9 .LE. NOTP .LE. 99 (AFTER
C                 TRANSPOSITION).  IF  NOCHROM ,  NOTP  GIVES THE
C                 NUMBER OF THE STAFF LINE OR SPACE THE NOTE SHOULD
C                 APPEAR ON, WITH 29=MIDDLE C (REGARDLESS OF CLEF).
C                 HENCE, IN THIS CASE THE PIANO'S LOW A IS 6 AND ITS
C                 TOP C IS 57.
C  JQP      I3    UNDER  CHROM ,  A VALUE OF 1 INDICATES QUARTER-TONE
C                 DOWN, 2 QUARTER-TONE UP FROM THE PITCH GIVEN BY
C                 NOTP .  IF  NOCHROM , THE ONE'S DIGIT IS AS JUST
C                 DESCRIBED, WHILE TEN'S DIGIT=1 MEANS DOUBLE FLAT, 2
C                 MEANS FLAT, 3 MEANS NATURAL, 4 SHARP, AND 5 DOUBLE
C                 SHARP.
C  NDURB    I3    DURATION.  IF  NDURB  AND  IDT  ARE BOTH ZERO, THE
C                 DURATION OF THE PREVIOUS NOTE READ IS REPEATED.
C                 OTHERWISE, THE MEANING OF  NDURB  AND  IDT DEPENDS
C                 ON WHETHER  NORHDEC  OR  RHDEC  IS IN EFFECT.  WITH
C                 NORHDEC , NDURB  IS THE RECIPROCAL OF THE BASIC DUR-
C                 ATION AND  IDT  THE NUMBER OF DOTS.  WITH  RHDEC ,
C                 NDURB  IS A NO. OF 128TH NOTES AND  IDT  A NO. OF
C                 WHOLE NOTES TO ADD TOGETHER TO DETERMINE THE DUR-
C                 ATION.  BASIC DURATIONS ARE LISTED WITH THEIR EQUIV-
C                 ALENTS BELOW.
C                                 ---WITH  NORHDEC---   WITH RHDEC
C                  NOTE VALUE      NDURB   MAX. OF IDT   NDURB  IDT
C
C                  DOUBLE LONG       -8         10           0    8
C                  LONG              -4          9           0    4
C                  BREVE             -2          8           0    2
C                  WHOLE (SEMIBREVE)  1          7           0    1
C                  HALF               2          6          64    0
C                  QUARTER            4          5          32    0
C                  EIGHTH             8          4          16    0
C                  16TH              16          3           8    0
C                  32ND              32          2           4    0
C                  64TH              64          1           2    0
C                  128TH            128          0           1    0
C
C  IDT      I3    SEE EXPLANATION OF  NDURB  ABOVE.
C  KSYM     5A2   EACH OF THE FIVE FIELDS INDICATES A SYMBOL ASSOC-
C                 IATED WITH THE NOTE.  ALL ES'S FOR A NOTE SHOULD
C                 APPEAR FIRST, THEN ALL SL'S, SU'S, AND SD'S, THEN
C                 A TI (IF PRESENT), THEN EVERYTHING ELSE.  ES  AND
C                 SL  CANNOT BOTH APPEAR ON A DECOMPOSED NOTE.  IN
C                 TERMS OF POSITIONING, SYMBOLS FALL INTO FOUR CAT-
C                 EGORIES - THOSE THAT ARE ALIGNED WITH THE NOTEHEAD
C                 AND ALWAYS GO ABOVE THE STAFF (MARKED 'A' BELOW),
C                 THOSE THAT ARE ALIGNED WITH THE NOTEHEAD AND CAN GO
```

```
C                   ABOVE OR BELOW THE HEAD OR STEM (NO MARKING), THOSE
C                   THAT FOLLOW THE NOTE (MARKED 'F'), AND MISCELLANEOUS
C                   UNALIGNED SYMBOLS (MARKED 'M').  THOSE IN THE 2ND
C                   CATEGORY ARE DRAWN WITH THE 1ST INNERMOST (CLOSEST
C                   TO THE NOTEHEAD), THEN THE 2ND, ETC.
C
C                   CODE  CAT.  MEANING
C
C                     ES        END SLUR.  UP TO 3 SLURS (AND TIES) MAY
C                               BE NESTED.
C                     SL        BEGIN SLUR.
C                     SU        BEGIN SLUR CURVING UP WITH CURVATURE
C                               GIVEN BY THE LEFTMOST DIGIT OF THE NEXT
C                               FIELD.
C                     SD        BEGIN SLUR CURVING DOWN WITH CURVATURE
C                               GIVEN BY THE LEFTMOST DIGIT OF THE NEXT
C                               FIELD.
C                     TI        TIE TO FOLLOWING NOTE.
C                     0         INDICATES NUMBER ZERO.
C                     1         NUMBER '1'.
C                     2         NUMBER '2'.
C                     3         NUMBER '3'.
C                     4         NUMBER '4'.
C                     5         NUMBER '5'.
C                     A         AN ACCENT MARK.
C                     AH        A HEAVY ACCENT (LIKE FRENCH CIRCUM-
C                               FLEX).
C                     .         STACCATO DOT.
C                     -         TENUTO LINE.
C                     +         A '+'.
C                     V     A   A 'V' (UPBOW SIGN).
C                     D     A   A DOWNBOW SIGN.
C                     FM    A   FERMATA (HOLD)
C                     TR    A   LETTERS 'TR'.
C                     TW    A   A WIGGLY LINE (BAROQUE TRILL).
C                     MR    A   MORDENT.
C                     ,     F   BREATH MARK AFTER NOTE.
C                     //    F   PAUSE MARK AFTER NOTE.
C                     TM    M   TREMOLO.  WHEN A FIELD CONTAINS  TM ,
C                               THE SUCCEEDING FIELD MUST INDICATE THE
C                               RHYTHM OF THE TREMOLO.  THIS IS THE
C                               LEFT-JUSTIFIED NO. OF SLASHES REQUIRED,
C                               SO THAT DIVIDING A NON-DOTTED NOTE
C                               INTO 8 WOULD BE INDICATED BY '3 '.  TM
C                               SHOULD NOT APPEAR MORE THAN ONCE ON A
C                               NOTE.
C                     GL    F   GLISSANDO.  THE NEXT FIELD MUST CON-
C                               TAIN THE SEMITONE PITCH TO BE SLID TO,
C                               THE NEXT FIELD AFTER THAT THE QUARTER-
C                               TONE INDICATOR.
C                     BM    M   BEGIN BEAMING HERE.
C                     EB    M   END BEAMING HERE.
C                   REGARDLESS OF WHETHER AUTO-BEAMING IS IN EFFECT
C                   SEE THE '* BEAM' COMMAND), SMUT  ATTEMPTS TO BEAM
C                   THE NOTES AND GRACE NOTES FROM THE ONE WITH 'BM'
```

```
C              THROUGH THE ONE WITH 'EB' WITHOUT BREAKS DUE TO
C              METER.  HOWEVER, THE 'BM' AND 'EB' MUST BOTH BE
C              WITHIN THE SAME BAR - BEAMING ACROSS BARLINES IS
C              NOT POSSIBLE.
C                H3    M     SQUARE NOTEHEAD.
C                H4    M     DIAMOND-SHAPED NOTEHEAD.
C                H6    M     'X' SHAPED NOTEHEAD.
C                H7    M     OMIT NOTEHEAD ENTIRELY.  IF THE NOTE
C                            HAS ACCIDENTALS OR DOTS, THEY ARE NOT
C                            OMITTED.
C               (BLANK)      INDICATES NO SYMBOL.
C              N.B.   , AND //  SHOULD NOT BE SPECIFIED WITH THE
C              SAME NOTE.
C
C  ** GRACE NOTE (G) COMMAND **
C  NOTP     I3   SAME AS IN NOTE COMMANDS.
C  JGP      I3   SAME AS IN NOTE COMMANDS.
C  NDURG    I3   SAME AS IN NOTE COMMANDS, EXCEPT THE LONGEST LEGAL
C                DURATION IS A HALF NOTE.
C  ISHG     I3   (NOT USED IN THIS VERSION.)
C
C  ** REST (R) COMMAND **
C  NOTP     I3   -99 MEANS 'INVISIBLE REST'.  OTHERWISE, NOTP  IS
C                THE NUMBER OF HALF-SPACES BY WHICH THE REST SHOULD
C                BE MOVED FROM ITS NORMAL POSITION, E.G. -6 WOULD
C                LEAVE A WHOLE REST HANGING FROM THE BOTTOM LINE.
C                ODD VALUES ARE NOT RECOMMENDED, SINCE THEY COULD
C                MAKE WHOLE RESTS LOOK LIKE HALF RESTS AND VICE
C                VERSA.
C  NDURB    I3   SAME AS IN NOTE COMMANDS, EXCEPT THAT -1 MEANS A
C                CENTERED WHOLE REST FILLING A BAR, REGARDLESS OF
C                METER, IF USED AT THE BEGINNING OF A BAR--ELSE IT
C                CALLS FOR WHATEVER RESTS ARE NEEDED TO COMPLETE
C                THE BAR.  ALSO  -N , FOR N .GT. 1, MEANS A 'MULTI-
C                BAR' REST OF  N  BARS DURATION--A LOGICAL BARLINE
C                SHOULD APPEAR ON EACH SIDE OF A MULTIBAR REST WITH
C                NOTHING IN BETWEEN.  IF DOING A SCORE, IT GEN-
C                ERATES N WHOLE RESTS SEPARATED BY BARLINES;  IF
C                DOING PARTS, IT GENERATES A THICK HORIZONTAL BAR
C                WITH THE NUMBER  N  ABOVE IT.
C  IDT      I3   SAME AS IN NOTE COMMANDS.  IGNORED WITH MULTIBAR
C                RESTS.
C                NDURB  AND  IDT  BOTH ZERO CAUSES THE DURATION OF
C                THE LAST REST (NOT NOTE) TO BE USED.
C  KRSYM    A2   A SYMBOL ASSOCIATED WITH THE REST.  THOSE MARKED
C                'F' BELOW FOLLOW THE REST, THOSE MARKED 'A' ARE
C                ALIGNED WITH IT AND ABOVE THE STAFF (THESE SHOULD
C                NOT BE USED WITH MULTIBAR RESTS).
C
C                 CODE  CAT.  MEANING
C
C                  FM    A    FERMATA (HOLD)
C                  ,     F    BREATH MARK AFTER REST
C                  //    F    PAUSE MARK AFTER REST
C
```

```
C   ** BARLINE (B) COMMAND **
C   KB       I3    IF  KB .LT. 0, THE TYPE OF BARLINE SPECIFIED BY
C                  THE VALUE OF  KB  IS DRAWN BUT THE BAR IS NOT
C                  LOGICALLY ENDED.  IF  KB .GE. 0 AND AUTOMATIC BAR-
C                  RING IS IN EFFECT, THE VALUE OF  KB  SPECIFIES THE
C                  TYPE OF THE NEXT AUTOMATICALLY GENERATED BARLINE.
C                  IF  KB .GE. 0 AND AUTO-BARRING IS NOT IN EFFECT,
C                  THE TYPE OF BARLINE SPECIFIED IS DRAWN AND THE BAR
C                  IS ENDED.
C                  VALUES HAVE THESE MEANINGS...
C                    1 MEANS SINGLE BAR,
C                    2 LIGHT DOUBLE BAR,
C                    3 HEAVY DOUBLE BAR,
C                    4 HEAVY DOUBLE BAR WITH REPEAT SIGN BEFORE IT,
C                    5 HEAVY DOUBLE BAR WITH REPEAT SIGN AFTER IT,
C                    6 HEAVY DOUBLE BAR WITH REPEAT SIGN BEFORE+AFTER,
C                    7 DOTTED BARLINE,
C                    8 SHORT, LIGHT SINGLE BAR,
C                    0 'INVISIBLE BAPLINE' (ACTS LOGICALLY LIKE A
C                      3ARLINE, FLUSHES BUFFERS, ETC.)
C                    N.B. IF DIFFERENT VOICES IN A CONNECTED GROUP
C                    HAVE A DIFFERENT BARLINE TYPE, PECULIAR RESULTS
C                    ARE LIKELY.
C
C   ** PERFORMANCE DIRECTION (P) COMMAND **
C   IPARM1   I1    .EQ. 1 INDICATES CONDITIONAL PERFORMANCE DIRECTION,
C                  WRITTEN ONLY IF THIS VOICE IS THE TOP OR ONLY
C                  VOICE IN A GROUP (SEE INITIALIZATION COMMANDS).
C                  .EQ. 0 INDICATES PERFORMANCE DIRECTION ALWAYS TO
C                  BE WRITTEN.
C   IPARM2   I2    ABSOLUTE VALUES 1 TO 5 INDICATE VERY SMALL TO VERY
C                  LARGE CHARACTERS.  IF  IPARM2  IS POSITIVE THEY
C                  WILL BE WRITTEN ABOVE THE STAFF, IF NEGATIVE BELOW.
C                  SIZE 3 IS BEST FOR MOST PURPOSES EXCEPT DYNAMIC
C                  LEVELS ('MP', 'FF', ETC.), FOR WHICH SIZE 4 (AND
C                  FONT 1) IS SUITABLE.  IF DOING A SCORE, CONDITION-
C                  AL PERFORMANCE DIRECTION SIZES WILL BE BOOSTED BY
C                  ONE.
C   IPARM3   I3    THE Y-DISPLACEMENT FROM THE NORMAL POSITION IN
C                  HALF-SPACES.  THE NORMAL POSITION IS 5 HALF-SPACES
C                  ABOVE THE STAFF OR 9 HALF-SPACES BELOW.
C   IPARM4   I1    FONT (0=NORMAL, 1=DYNAMICS FONT, 2=ITALICS).  N.B.
C                  ON CDC COMPUTERS, FONT 1 ASSUMES ALL ALPHABETIC
C                  CHARACTERS ARE UPPER CASE AND FORCES THEM INTO
C                  LOWER CASE.
C   IPARM5   I2    THE NUMBER OF CHARACTERS TO BE WRITTEN (MAX. 48).
C   ---      12A4  THE LITERAL CHARACTERS TO BE WRITTEN, STARTING
C                  ABOVE OR BELOW THE NEXT NOTE OR REST.
C
C   ** ARTIFICIAL GROUP OR GROUPET (A) COMMAND **
C   NAG      I3    THE GROUP'S TOTAL DURATION IN TERMS OF ITS BASIC
C                  UNIT, ALSO TAKEN AS THE ACCESSORY NUMERAL TO BE
C                  PRINTED.
C   JGP      I3    THE GROUP'S DURATIONAL UNIT, INDICATED IN THE SAME
C                  WAY AS  NDURB  FOR NOTES AND RESTS, EXCEPT THAT IT
C                  MUST BE .GE. 1 WITH  NORHDEC  AND SHOULD BE .LE.
```

```
C                   128 AND A POWER OF 2 WITH  RHDEC .
C   MOVNAG    I3    .LE. -99 MEANS DO NOT PRINT ACCESSORY NUMERAL.
C                   OTHERWISE  MOVNAG  IS THE Y-DISPLACEMENT FOR THE
C                   ACCESSORY NUMERAL, IN HALF-SPACES.
C                   IN SIMPLE METERS, DURATIONS OF NOTES IN ARTIFICIAL
C                   GROUPS ARE ALWAYS REDUCED.  IN COMPOUND METERS,
C                   THIS DEPENDS ON THE CONTROL COMMANDS  SHRINK  AND
C                   NOSHRINK (WHICH SEE).  N.B. IF  DESYNC  IS ON, NO
C                   ARTIFICIAL GROUP STARTING OFF THE BEAT MAY CROSS
C                   THE FOLLOWING BEAT.  SEE THE DESCRIPTION OF THE
C                   'S' COMMAND BELOW.
C
C   ** STAFF-CLEF-KEY SIGNATURE-METER (S) COMMAND **
C                   THE FIRST 'S' COMMAND FOR EACH VOICE SHOULD SPECIFY
C                   CLEF AND TIME SIGNATURE.  AN INITIAL KEY SIGNATURE
C                   OF NO SHARPS OR FLATS IS ASSUMED.
C   KIND      I3    IF KIND .NE. 0, START A NEW SYSTEM.  KIND .GT. 0
C                   MEANS RIGHT-JUSTIFY THE OLD SYSTEM, .LT. 0 MEANS
C                   DO NOT.  +OR- 1 IMPLIES A CONTINUATION OF THE SAME
C                   PIECE, SO THE OLD KEY SIGNATURE WILL BE REPEATED
C                   AND/OR CANCELLED AND MEASURE NUMBERS WILL CONTINUE.
C                   +OR- 2 IMPLIES STARTING A NEW PIECE, SO THE OLD KEY
C                   SIGNATURE WILL BE FORGOTTEN.  MEASURE NOS. WILL
C                   RESTART AT 1, ETC.  +OR- 3 IS THE SAME AS +OR- 2,
C                   EXCEPT IT ALSO MEANS START A NEW PAGE.
C                   KIND=0 MEANS DO NOTHING.
C                   N.B. KIND = +OR- 2 OR 3 SHOULD NEVER BE USED IN
C                   SCORES.  WHEN DOING A SCORE, IF ANY VOICE REQUESTS
C                   A NEW SYSTEM (I.E., HAS KIND .NE. 0) ONE WILL BE
C                   STARTED.  TWO DIFFERENT NONZERO VALUES OF  KIND
C                   IN THE SAME BAR IN DIFFERENT VOICES SHOULD NEVER
C                   BE USED.  STARTING A NEW SYSTEM ALWAYS IMPLIES
C                   ENDING THE CURRENT BAR.
C   NUCLEF    I3    INDICATES NEW CLEF AS FOLLOWS - 1 TREBLE, 2 SO-
C                   PRANO, 3 MEZZO-SOPRANO, 4 ALTO, 5 TENOR, 6 BAR-
C                   ITONE, 7 BASS.  0 MEANS DO NOT CHANGE CLEFS.
C   KSAX      I3    NEW KEY SIGNATURE, AS FOLLOWS -
C                   -7 .LE. KSAX .LT. 0 MEANS CHANGE TO -KSAX FLATS,
C                   KSAX=0 MEANS NO CHANGE OF KEY SIGNATURE,
C                   0 .LT. KSAX .LE. 7 MEANS CHANGE TO KSAX SHARPS,
C                   KSAX .GT. 7 MEANS CHANGE TO 0 SHARPS AND FLATS.
C                   FLAT AND SHARP KEY SIGNATURES RESPECTIVELY INVOKE
C                   '*  FLAT' AND '*  NOFLAT'  CONTROL COMMANDS.
C   MN        I3    THE NUMERATOR OF THE NEW TIME SIGNATURE.  IF MN=0,
C                   THE TIME SIG. DOES NOT CHANGE AND MD AND MSPEC ARE
C                   IGNORED.
C   MD        I3    THE DENOMINATOR OF THE TIME SIGNATURE.
C                   IF  MN  IS GREATER THAN 3 AND A MULTIPLE OF 3 (6,
C                   9, 12, ETC.), THE METER IS CONSIDERED COMPOUND,
C                   OTHERWISE IT IS CONSIDERED SIMPLE.  IN COMPOUND
C                   METERS, 3 OF THE UNITS SPECIFIED BY  MD  MAKE A
C                   BEAT, IN SIMPLE METERS ONE SUCH UNIT MAKES A BEAT.
C   MSPEC     I3    IF  MSPEC .NE. 0, THE LOGICAL TIME SIGNATURE IS
C                   STILL GIVEN BY  MN/MD , BUT WHAT IS WRITTEN ON THE
C                   MUSIC IS--
```

```
C                 IF MSPEC .GT. 0, THEN  MSPEC ,
C                 IF MSPEC .EQ. -1, A SLASHED 'C',
C                          .EQ. -2, A 'C',
C                          .EQ. -3, NOTHING (BUT SPACE IS LEFT),
C                          .EQ. -4, NOTHING AND NO SPACE IS LEFT.
C  MOVEY    I3  Y-DISPLACEMENT FROM THE NORMAL POSITION OF THE
C               ENTIRE SYSTEM, IN HALF-SPACES.  IGNORED IF KIND=0.
C               THIS IS INTENDED FOR USE MOSTLY WITH SINGLE LINE
C               MUSIC.  IF, IN A SCORE, SEVERAL VOICES HAVE DIFF-
C               ERENT NONZERO VALUES, STRANGE RESULTS ARE LIKELY.
C
C  ** IDENTIFICATION/INDENT (I) COMMAND **
C  ISIZE    I1  SIZE OF CHARACTERS, 1 (VERY SMALL) TO 5 (VERY
C               LARGE).  (DEFAULT=2 IF STAFF IS SHARED, OTHERWISE
C               3.)
C  MOVEY    I3  Y-DISPLACEMENT FROM THE NORMAL POSITION, IN HALF-
C               SPACES.  THE NORMAL POSITION IS, IF AN UNSHARED
C               STAFF, THE CENTER OF THE STAFF, OR IF A SHARED
C               STAFF, JUST ABOVE OR BELOW THE CENTER.
C  IFONT    I1  FONT.  0=NORMAL, 1=DYNAMICS FONT, 2=ITALICS.  (SEE
C               NOTE ON DYNAMICS FONT UNDER THE 'P' COMMAND.)
C  MARG     A1  'M'=LEFT MARGIN CHANGE ('HANGING INDENT'), ANY-
C               THING ELSE=INDENT RELATIVE TO CURRENT LEFT MARGIN
C               FOR THIS STAFF ONLY.
C  INDENT   I2  DISTANCE TO INDENT STAFF IN CHARACTERS, .GE. 0.
C  NOTP     I2  NUMBER OF CHARACTERS IN THE IDENTIFICATION (MAX-
C               IMUM 32).
C  ISTRNG  8A4  LITERAL CHARACTERS TO BE WRITTEN AT THE LEFT END
C               OF THE CURRENT STAFF.  IF  INDENT=0 THE STAFF WILL
C               BE INDENTED BY SLIGHTLY MORE THAN  NOTP  CHAR-
C               ACTERS.  N.B.  FOR PROPER RESULTS, THIS COMMAND
C               SHOULD BE GIVEN IN THE FIRST MEASURE AFTER AN 'S'
C               COMMAND THAT STARTS A NEW SYSTEM.  IF MORE THAN
C               ONE 'I' COMMAND APPLIES TO A SINGLE STAFF, THE
C               LAST ONE WILL BE USED.
C
C  ** CONTROL (*) COMMAND **
C KTLV     I3   IN  BEAM, CLEF, TRANSP, RHYTH, ECHO, SET, IFEQ,
C               SHARE, TNOW, TLFT, HCROWD, HBASE, HMASH, JUST,
C               REPKS COMMANDS, AS DESCRIBED BELOW.  ON ALL OTHER
C               COMMANDS, IGNORED.
C  KTLWD  2A4   THE CONTROL WORD, LEFT JUSTIFIED.  ONLY THE FIRST
C               FOUR CHARACTERS ARE SIGNIFICANT, E.G. 'CHRONIC' IS
C               TAKEN TO MEAN 'CHROM'.  CONTROL WORDS ARE ($ IND-
C               ICATES DEFAULT CONDITIONS, ASSUMED AT THE BEGIN-
C               NING OF EACH VOICE)--
C               $ DESYNC    DECOMPOSE (INTO TWO OR MORE TIED
C                           NOTES) ALL NOTES THAT WOULD OBSCURE
C                           THE UNDERLYING RHYTHM OR CROSS THE
C                           BARLINE.  THIS ASSUMES NO MORE THAN 10
C                           BEATS PER BAR (NOTE THAT 12/8 IS OK,
C                           SINCE IT'S 4 BEATS PER BAR).
C                 NODESYNC  DO NOT DECOMPOSE NOTES.
C                 BEAM      BEAMING CONTROL, CONSISTING OF EXTENT
C                           OF EACH BEAMABLE AREA (THE TEN'S DIG-
C                           IT) AND AGGRESSIVENESS OF BEAMING (THE
C                             ONE'S DIGIT).  EXTENT=0 MEANS EACH BEAM-
C                             ABLE AREA EXTENDS FOR ONE BEAT (AS DE-
C                             TERMINED FROM THE TIME SIGNATURE),
C                             =1 MEANS FOR ONE UNIT OF RHYTHMIC
C                             STRENGTH ONE LEVEL ABOVE THE BEAT,
C                             =2 FOR ONE UNIT OF RHYTHMIC STRENGTH
C                             TWO LEVELS ABOVE THE BEAT (ALMOST AL-
C                             WAYS THE WHOLE BAR).  FOR EXAMPLE, IN
C                             5/4 TIME, EXTENT=0 MEANS QUARTER-NOTE
C                             DURATION, =1 ALTERNATE HALF AND DOTTED-
C                             HALF DURATION, =2 THE WHOLE BAR.  WITH
C                             AGGRESSIVENESS=1, THESE WOULD BE WRIT-
C                             TEN '* 01BEAM', '* 11BEAM', AND
C                             '* 21BEAM', RESPECTIVELY.  IF AGGRESS-
C                             IVENESS=0, NO BEAMING IS DONE, RE-
C                             GARDLESS OF EXTENT.  AGRESSIVENESS=1
C                             MEANS TIMID BEAMING, I.E. ALL SUITABLE
C                             GROUPS OF NOTES AND OF GRACE NOTES
C                             WITHIN EACH BEAMABLE AREA ARE BEAMED
C                             TOGETHER BUT BEAMS ARE BROKEN BY ANY
C                             SYMBOL THAT MIGHT CAUSE INTERFERENCE
C                             (RESTS, CLEF CHANGES, ETC.).  AGRESS-
C                             IVENESS=2 MEANS BEAMS WITHIN AN AREA
C                             ARE BROKEN ONLY WHEN ABSOLUTELY NEC-
C                             ESSARY--SOMETIMES NOT EVEN THEN.  (DE-
C                             FAULT=01, I.E. 1-BEAT AREAS AND TIMID)
C                  $ BAR      AUTOMATICALLY (ACCORDING TO TIME SIG-
C                             NATURE) SUPPLY SINGLE BARLINES.
C                    NOBAR    DO NOT SUPPLY BARLINES.
C                    CLEF     CHANGE TO THE BETTER CLEF OF THE TWO
C                             GIVEN IN THE FIRST TWO DIGITS OF  KTLV
C                             IF THE CURRENT CLEF REQUIRES TOO MANY
C                             LEDGER LINES.  CODES FOR THE CLEFS ARE
C                             THE SAME AS ON THE 'S' COMMAND.  FOR
C                             EXAMPLE, '*57 CLEF' MEANS USE EITHER
C                             TENOR OR BASS CLEF.
C                  $ NOCLEF   DO NOT CHANGE CLEFS EXCEPT AS SPECIFIED
C                             BY S-C-K-M COMMANDS.
C                    TRANSP   TRANSPOSE EVERYTHING  KTLV  SEMITONES.
C                  $ FLAT     USE FLATS FOR ALL HALF-STEP ACCIDENT-
C                             ALS.
C                    NOFLAT   USE SHARPS.
C                  FLAT  AND  NOFLAT  ARE AUTOMATICALLY INVOKED BY
C                  FLAT AND SHARP KEY SIGNATURES, RESPECTIVELY.  IF
C                  NOCHROM  IS IN EFFECT, THEY ARE IGNORED.
C                    SHRINK   IN COMPOUND METERS, COMPRESS DURATIONS
C                             WITHIN ARTIFICIAL GROUPS.
C                  $ NOSHRINK IN COMPOUND METERS, STRETCH DURATIONS
C                             WITHIN ARTIFICIAL GROUPS.
C                             DURATIONS OF NOTES IN ARTIFICIAL
C                             GROUPS IN SIMPLE METERS ARE ALWAYS
C                             COMPRESSED.
C                    RHYTH*   MULTIPLY THE DURATIONS OF NOTES AND
C                             RESTS AND DIVIDE DENOMINATORS OF TIME
C                             SIGNATURES BY  KTLV (WHICH MUST BE 1,
C                             2, OR 4).
```

```
C                      RHYTH/      DIVIDE DURATIONS AND MULTIPLY DENOMI-
C                                  NATORS BY  KTLV (WHICH MUST BE 1, 2,
C                                  OR 4).
C                      RHDEC       ACCEPT RHYTHMIC INPUT IN 'DECIMAL'
C                                  FORM.  N.B. RHDEC  WILL NOT WORK WITH
C                                  NODESYNC .
C                    $ NORHDEC     ACCEPT RHYTHMIC INPUT AS RECIPROCALS
C                                  OF ACTUAL DURATIONS.
C                    $ CHROM       ACCEPT PITCH INPUT IN CHROMATIC FORM.
C                      NOCHROM     ACCEPT PITCH INPUT IN NONCHROMATIC
C                                  FORM.
C                      ECHO        READ (E.G. SCAN FOR DATA) THE NEXT
C                                  LINE  KTLV  TIMES.  N.B. THE LINE
C                                  ECHOED MAY NOT CONTAIN AN 'X' COMMAND.
C                      READALT     IF CURRENTLY READING FROM THE STANDARD
C                                  READ UNIT, SWITCH TO THE ALTERNATE.  IF
C                                  CURRENTLY READING FROM THE ALTERNATE
C                                  READ UNIT, SWITCH TO THE STANDARD.
C                      SET         SET VARIABLE FOR TESTING BY  IFEQ  TO
C                                  KTLV (DEFAULT=999).
C                      IFEQ        IF  KTLV .NE. THE SET VARIABLE, SKIP
C                                  INPUT UNTIL AN  ENDIF  IS FOUND.  EX-
C                                  CEPTION...IF THE SET VARIABLE=999,
C                                  NEVER SKIP.
C                      ENDIF       TERMINATE THE RANGE OF AN  IFEQ .
C.                     SHARE       OVERRIDE SHARED STAFF STATUS IN PASS I
C                                  ONLY.  KTLV=1 TELLS PASS I THIS IS THE
C                                  UPPER VOICE OF A SHARED STAFF, =-1
C                                  THIS IS THE LOWER, =0 THIS STAFF IS
C                                  UNSHARED.
C                    IF EITHER THE TNOW OR THE TLFT COMMAND IS USED TO
C                    DECREASE THE ELAPSED TIME IN A MEASURE, TERRIBLE
C                    RESULTS MAY OCCUR.  THEY ARE BEST USED ONLY FOR
C                    ANACRUSES.
C                      TNOW        SET ELAPSED TIME WITHIN CURRENT BAR TO
C                                  (KTLV/NUMBER FOLLOWING 'TNOW' IN I3
C                                  FORMAT).
C                      TLFT        SET REMAINING TIME WITHIN CURRENT BAR
C                                  TO (KTLV/NUMBER FOLLOWING 'TLFT' IN I3
C                                  FORMAT).
C                    DEFAULTS FOR THE FOLLOWING CONTROL WORDS ARE SET
C                    ONLY AT THE VERY BEGINNING OF THE RUN, AND WHAT-
C                    EVER VALUE IS IN EFFECT AT THE END OF THE DATA IS
C                    TAKEN TO APPLY THROUGHOUT.  IF IT IS NECESSARY TO
C                    VARY THEM WITHIN A RUN, A  USER3  OR  USER4  CAN
C                    DO SO.
C                    $ SYST        WHEN A SYSTEM IS FILLED, AUTOMATICALLY
C                                  BEGIN ANOTHER WITH THE SAME CLEFS AND
C                                  KEY SIGNATURES.
C                      NOSYST      DO NOT BEGIN A NEW SYSTEM UNTIL AN
C                                  S-C-K-M COMMAND WITH KIND .NE. 0 IS
C                                  FOUND.
C                      HCROWD      SET THE 'IDEAL' HORIZONTAL SPACING TO
C                                  NOTP PERCENT OF THE USUAL VALUE.  IF
C                                  THE SHORTEST NOTE IN THE SCORE IS, SAY,
C                                  A QUARTER NOTE, ONE MIGHT SET HCROWD
```

```
C                           TO 75 TO AVOID LEAVING SPACE FOR
C                           SHORTER NOTES THAT NEVER OCCUR.
C                 HBASE     SET THE BASE FOR 'IDEAL' HORIZONTAL
C                           SPACING CALCULATION TO  1+(HBASE/1000).
C                           HBASE=0 GIVES CONSTANT SPACING, IN-
C                           DEPENDENT OF DURATION.  HBASE=999
C                           GIVES APPROXIMATELY LINEAR SPACING,
C                           PROPORTIONAL TO DURATION.  BOTH LOOK
C                           BAD.  REASONABLE VALUES ARE FROM ABOUT
C                           350 TO 700.  (DEFAULT=520)
C               TO CLARIFY THE ABOVE TWO COMMANDS, 'IDEAL' SPAC-
C               ING IS CALCULATED BY THE EXPRESSION
C                   CROWD*BASE**(LOG2(DURATION)) ,
C               WHERE  CROWD=HCROWD/100  AND  BASE=1+(HBASE/1000) .
C               THIS IS IDEAL, NOT ACTUAL, SPACE BECAUSE EXTRA
C               SPACE IS LEFT WHERE NEEDED FOR ACCIDENTALS, AUG-
C               MENTATION DOTS, ETC., AND THEN RIGHT-JUSTIFICA-
C               TION USUALLY AFFECTS SPACING FURTHER.
C                 HMASH     SQUEEZE NOTES (INCLUDING ASSOCIATED
C                           ACCIDENTALS AND DOTS) TO  NOTP  PER-
C                           CENT OF THEIR WIDTH.  THIS IS DIFF-
C                           ERENTR FROM AND, IN A WAY, MORE
C                           DRASTIC THAN CHANGING  HCROWD  AND
C                           HBASE , WHICH AFFECT ALL SYMBOLS,
C                           BUT ONLY THEIR POSITIONS, NOT THEIR
C                           SHAPES.
C               $ MEASNO    PUT MEASURE NUMBERS AT THE LEFT END
C                           OF EACH SYSTEM.
C                 NOMEASNO  DO NOT PUT MEASURE NUMBERS.
C                 JUST      KTLV=0 MEANS DO NOT RIGHT-JUSTIFY ANY-
C                           THING, =1 MEANS RIGHT-JUSTIFY EVERY-
C                           THING EXCEPT THE LAST SYSTEM (OR LAST
C                           STAFF OF EACH PART, IF DOING PARTS),
C                           =2 MEANS RIGHT-JUSTIFY EVERYTHING.
C                           (DEFAULT=2)
C                 REPKS     CONTROLS REPEATING THE OLD KEY SIGNA-
C                           TURE AT THE BEGINNING OF A STAFF IF IT
C                           IS CHANGED IN THE FIRST MEASURE OF THE
C                           STAFF.  KTLV=0 MEANS DO NOT REPEAT THE
C                           OLD KEY SIGNATURE, =1 MEANS DO.  NOTE
C                           THAT IF THE KEY SIGNATURE IS CHANGED
C                           IN THE MIDDLE OF THE BAR, =0 WILL PRO-
C                           DUCE INCORRECT NOTATION.  (DEFAULT=1)
C
C  ** EDIT (E) COMMAND **
C  USING EDIT COMMANDS REQUIRES A KNOWLEDGE OF  SMUT  PASS4 COM-
C  MAND FORMATS.  SEE ALL OF PASS4 SOURCE CODE.  THERE ARE FIVE
C  SUBCOMMANDS...
C    COMMAND   FORMAT
C    -------   ------
C    CHANGE    CRWWFNNNNN
C    INSERT    IRII XXXXX OO PP NNNNN...AA...  (N'S AND A'S 9I5,6A2)
C    AGAIN     AR
C    DELETE    DR
C    PRINT     PR
C
```

```
C     THE OVERALL FORMAT IS
C         (A1, I1, I2, A1, I5, 2(1X, I2), 1X, 9I5, 3A2) .
C
C     THE R'S (RECORD NUMBERS TO EDIT) ARE RELATIVE TO THE NEXT
C       NONEDITOR COMMAND RECORD, E.G. R=2 MEANS THE RECORD AFTER
C       NEXT.
C     WW=WORD NO. OF RECORD TO CHANGE...-2=INSTRUMENT NO., -1=X,
C       0=OP CODE, .GT. 0=PARAMETER NO. WW.  WW .LT. -2=NO OP.
C     F='R' MEANS NNNNN IS RELATIVE TO THE OLD VALUE.
C     II=INSTRUMENT NO., XXXXX=X, OO=OPCODE, PP=NO. OF PARAMETERS.
C     NNNNN=INTEGER PARAMETER, AA=ALPHANUMERIC PARAMETER, EXCEPT IF
C       WW=-1 (ON THE CHANGE COMMAND) NNNNN/1000 IS USED.
C     RECORD NUMBERS ARE NOT AFFECTED BY INSERTS AND DELETES, E.G.
C       EVEN IF THE FIRST COMMAND IS 'D1' A 'C2' STILL REFERS TO
C       THE SECOND FOLLOWING RECORD OF THE FILE.  FOR A GIVEN REC-
C       ORD NUMBER, THE FOLLOWING RULES APPLY...
C        IF A DELETE COMMAND APPEARS, NOTHING ELSE IS ALLOWED.
C          THIS MEANS THAT ONE CANNOT REPLACE A RECORD BY DOING
C          'D' FOLLOWED BY 'I', BUT MUST USE 'I' ON THE PREVIOUS
C          RECORD, THEN 'D' ON THIS ONE.
C        ANY SEQUENCE OF 'C'S AND 'A'S IN ANY COMBINATION IS LEGAL
C          AND DOES WHAT YOU WOULD EXPECT.
C
C     RECORD NUMBERS AND PARAMETERS FOR EDITING CAN BE DETERMINED BY
C       1ST RUNNING  SMUT  WITH LISTC .GT. 0.  REGARDLESS OF LISTC,
C       SMUT WILL PRINT BACK RECORDS ALTERED BY 'C', 'A', OR 'I'
C       COMMANDS.  (SEE SUBROUTINE  PASS4 .)  NOTE THAT CHANGING
C       ALMOST ANYTHING IN THE REGULAR INPUT STREAM (E.G., ASKING
C       FOR SCORE INSTEAD OF PARTS OR VICE-VERSA, CHANGING THE PAGE
C       SIZE, ADDING OR DELETING NOTES OR RESTS, ETC.) MIGHT AFFECT
C       THE RECORD NUMBERS OR PARAMETERS, SO THE USER SHOULD BE
C       CAREFUL TO HAVE EVERYTHING ELSE SETTLED BEFORE MAKING UP
C       EDITOR COMMANDS.
C
C     COMMANDS TO THIS EDITOR EFFECTIVELY BYPASS THE ERROR CHECKING
C       DONE BY PASSES 1, 2, AND 3 OF  SMUT , SO A MISTAKEN EDITOR
C       COMMAND CAN CAUSE JUST ABOUT ANYTHING.
C
C   ** MISCELLANEOUS SYMBOL (M) COMMAND **
C   MOVE     I3   LEAVE  (MOVE/8)*(HEIGHT OF STAFF) SPACE BEFORE
C                 SYMBOL.
C   KSYM     A2   ONE OF -
C
C                   ,    BREATH MARK
C                   //   PAUSE MARK
C                   (BLANK) NONE
C
C   ** CALL USER ROUTINE (U) COMMAND **
C   IPARM1   I3   1, 3, OR 4 RESPECTIVELY MEAN CALL USER-SUPPLIED
C                 SUBROUTINE  USER1 , USER3 , OR  USER4  IN PASS I,
C                 III, OR IV AT THIS POINT IN PROCESSING THE DATA.
C                 THESE ROUTINES CAN THEN PERFORM ANY FUNCTIONS THEY
C                 WISH.  THIS FEATURE ADDS CONSIDERABLE POWER TO
C                 SMUT .   HOWEVER, WRITING A USER ROUTINE REQUIRES
```

```
C                   A FAIRLY DETAILED KNOWLEDGE OF SMUT'S INTERNAL
C                   WORKINGS.
C   IPARM2-4 I3,I4,I3  PARAMETERS TO BE PASSED TO THE USER ROUTINE.
C
C   ** OCTAVA (T) COMMAND **
C   NT        I3    +OR- 8, 15, OR 22 MEANS START OCTAVE SIGN NOTATION
C                   WITH NEXT NOTE--ABSOLUTE VALUE OF 8 MEANS 8VA, 15
C                   MEANS 15MA, 22 MEANS 22DA.  POSITIVE VALUE MEANS
C                   SOPRA, NEGATIVE BASSA.
C                   NT=0 OR +OR- 1 MEANS LOCO (END OCTAVE SIGN NOTA-
C                   TION) IMMEDIATELY.
C   MOVEY     I3    THE Y-DISPLACEMENT OF THE NUMBER AND DOTTED LINE
C                   IN HALF-SPACES, RELATIVE TO THE NORMAL POSITION.
C                   THE VALUE ON THE START OCTAVE NOTATION COMMAND IS
C                   USED AND THAT ON THE LOCO IS IGNORED.
C
C
C   ** END-OF-VOICE (X) COMMAND **
C   ---       A1    'X' (I.E., THE COMMAND 'XX') INDICATES END-OF-
C                   DATA.  ANYTHING ELSE MEANS SKIP TO THE NEXT LINE
C                   AND BEGIN READING DATA FOR THE NEXT VOICE.
C
```

PROGRAM OPERATION

USING SMUT MOST EFFECTIVELY REQUIRES SOME UNDERSTANDING OF ITS INTERNAL WORKINGS. SMUT 2.9 CONSISTS OF FOUR 'PASSES', AS FOLLOWS--

PASS I READS THE ENTIRE DATA FILE, ARRANGED AS ALL OF VOICE 1, THEN ALL OF VOICE 2, ETC., UNTIL A 'XX' COMMAND OR END-OF-FILE IS FOUND. DEPENDING ON WHAT OPTIONS ARE ENABLED BY CONTROL COMMANDS, IT PERFORMS VARIOUS 'HIGH-LEVEL' FUNCTIONS--DECIDING WHICH NOTES TO BEAM TOGETHER, WHEN TO CHANGE CLEFS, HOW TO SIMPLIFY ('DESYNCOPATE' OR CLARIFY) RHYTHM, APPROXIMATELY WHERE SLURS AND ACCESSORY NUMERALS SHOULD BE PLACED, WHAT COURTESY ACCIDENTALS TO USE, ETC.--AND WRITES ITS RESULTS ON A SCRATCH FILE (LOGICAL UNIT 10), STILL WITH ALL OF EACH VOICE TOGETHER. THE INFORMATION WRITTEN IS EXACTLY THE VSAME, WHETHER A SCORE OR SET OF PARTS IS CALLED FOR, WITH ONE EXCEPTION--MULTIBAR RESTS. IF PARTS ARE WANTED, THIS KIND OF 'R' COMMAND IS REPLACED BY WHOLE REST COMMANDS SEPARATED BY BARLINE COMMANDS, IF A SCORE, THE MULTIBAR REST COMMAND IS KEPT.

IN ORDER TO ALIGN EVERYTHING IN THE SCORE PROPERLY, IF ONE IS CALLED FOR, SMUT NEEDS TO EXAMINE EVERYTHING HAPPENING IN BAR 1 TOGETHER, THEN EVERYTHING IN BAR 2, ETC., SO IN THIS CASE PASS II SORTS THE SCRATCH FILE ONTO A SECOND SCRATCH FILE (UNIT 11). IF A SET OF PARTS IS WANTED, PASS II DOES ABSOLUTELY NOTHING EXCEPT PRINT SOME INFORMATION PRODUCED BY PASS I.

PASS III READS SCRATCH FILE 2, ONE BAR OF SCORE OR ONE BAR OF A PART AT A TIME, AND ASSIGNS Y-COORDINATES TO SEVERAL THINGS THAT COULD NOT BE HANDLED BY PASS I (FOR EXAMPLE, EXACT COORDINATES OF BEAMS AND SLURS). MORE IMPORTANTLY, IT ASSIGNS TEMPORARY X-COORDINATES TO EVERYTHING - I.E., IT JUSTIFIES THE BAR - AND IT DECIDES WHEN A SYSTEM IS FULL AND A NEW ONE MUST BE BEGUN. THIS IS DONE BY THE FOLLOWING METHOD...

1. MAKE A LIST OF ALL 'TIMES' WITHIN THE BAR WHERE ANYTHING HAPPENS, E.G. A NOTE OR REST IN ANY VOICE STARTS. (THIS LIST IS SOMETIMES CALLED THE 'RHYTHMIC SPINE'.)
2. ASSIGN 'IDEAL' SPACINGS TO EACH OF THESE TIMES WITH THE FORMULA IDEAL SPACE = 2.5*CROWD*(BASE**LOG2(TIME)) . CROWD AND BASE ARE SET AS DESCRIBED UNDER THE HCROWD AND HBASE COMMANDS, ABOVE. THE DEFAULTS ARE 1.0 AND 1.520.
3. IF THIS DOES NOT LEAVE ENOUGH SPACE BETWEEN CONSECUTIVE SYMBOLS WITHIN ANY VOICE, INCREASE IT AS NECESSARY.
4. IF THE RESULTING BAR WIDTH OVERFLOWS THE CURRENT STAFF AT ALL AND AUTOMATIC STAFF BREAKING WAS REQUESTED, START A NEW STAFF. EVEN IF NOT, IF IT OVERFLOWS THE CURRENT STAFF BADLY, GIVE AN ERROR MESSAGE AND START A NEW STAFF. IF STARTING A NEW STAFF RESULTS IN THE OLD ONE HAVING NO MUSIC IN IT, GIVE AN ERROR MESSAGE.

PASS III WRITES RECORDS EACH DESCRIBING ONE MUSICAL SYMBOL ON A THIRD SCRATCH FILE (UNIT 12), AND CONTROL INFORMATION ON A FOURTH (UNIT 13). THESE FILES ARE IMPORTANT IF A PASS IV EDITOR IS USED - SEE BELOW.

PASS IV READS SCRATCH FILE 3 AND DOES THE ACTUAL PLOTTING. AS IT GOES, IT MULTIPLIES THE TEMPORARY X-COORDINATES OF EACH SYMBOL BY AN APPROPRIATE CONSTANT SO THAT THE SYSTEM IS STRETCHED TO EXACTLY THE CORRECT LENGTH (THAT GIVEN ON THE LAYOUT LINE), UNLESS RIGHT-JUSTIFICATION HAS BEEN TURNED OFF GLOBALLY BY A '* JUST' COMMAND OR FOR THIS SYSTEM BY A 'S' COMMAND THAT ENDED IT AND BEGAN THE NEXT SYSTEM.

TWO EDITING VERSIONS OF THE PASS IV INPUT SUBROUTINE (READ4) EXIST. THESE ALLOW CHANGING OR DELETING ANYTHING ON UNIT 12 OR ADDING NEW RECORDS TO IT BEFORE ANYTHING IS ACTUALLY DRAWN. IF THIS IS DONE BY MAKING TWO SEPERATE RUNS OF SMUT IN EDIT MODE, BOTH PASS IV INPUT FILES - UNITS 12 AND 13 - MUST BE SAVED AFTER THE FIRST RUN AND AVAILABLE TO THE SECOND. THE SYSTEM FILE NAMES WILL BE IMPLEMENTATION-DEPENDENT, BUT ON CDC MACHINES THEY ARE 'ZSMUT3' AND 'ZSMUTX' RESPECTIVELY.

PRINTER LISTING

IN ADDITION TO PLOTTED OUTPUT, THE PROGRAM PRODUCES A PRINTER LISTING OF THE DATA PLUS COMMENTS AND ERROR MESSAGES. AMONG THE COMMENTS PRINTED ARE INDICATIONS OF WHERE EACH BAR AND EACH LINE OF PLOTTED MUSIC BEGINS. THE LISTING WILL BE SHORTENED CONSIDERABLY AND ALL ERROR MESSAGES WILL STILL APPEAR IF LISTC IS .LT. 0 (SEE LAYOUT LINE).

IN SOME CASES VARIABLE NAMES WILL APPEAR IN THE LISTING WITH THEIR CURRENT VALUES. BESIDES INPUT PARAMETERS AS DESCRIBED ABOVE, THESE INCLUDE TNOW (ELAPSED 'TIME' IN WHOLE NOTES SINCE THE BEGINNING OF THE BAR, E.G. 0.250 MEANS 1 QUARTER NOTE INTO THE BAR) AND XFACT (X-COORDINATE SCALING FACTOR, FOR RIGHT-JUSTIFYING THE SYSTEM).

ERROR MESSAGES

A TYPICAL SMUT ERROR MESSAGE LOOKS LIKE THIS...

```
N18******COL.21  AN OCTAVA OF   3 IS ALREADY IN EFFECT.
```

THE 'N' IN 'N18' INDICATES THIS IS A NONFATAL ERROR. OTHER TYPES OF ERRORS ARE 'F' (FATAL), 'W' (WARNING--MINOR PROGRAM ERROR OR POSSIBLE USER ERROR), AND 'S' (A SEVERE PROGRAM ERROR). 'COL. 21' INDICATES THIS ERROR WAS FOUND WHILE PROCESSING THE FIELD STARTING AT COLUMN 21 OF THE PRECEDING LINE. A COLUMN NUMBER IS GIVEN ONLY IF THE ERROR IS DETECTED DURING PASS I. IN MOST OTHER CASES, SMUT WILL GIVE THE BAR NUMBER AND, IF APPROPRIATE, THE VOICE NUMBER WHERE THE ERROR WAS DETECTED.

WHATEVER CAUSED AN 'N' ERROR WILL GENERALLY BE IGNORED (SUBJECT TO MAXERR --SEE THE DESCRIPTION OF THE LAYOUT LINE PARAMETER) AND EXECUTION WILL CONTINUE, WHILE THE OCCURENCE OF AN 'F' OR 'S' ERROR WILL CAUSE THE PROGRAM TO STOP IMMEDIATELY. IF AN 'S' ERROR

OCCURS, FIRST CHECK WHETHER THERE ARE ANY ERRORS IN THE DATA THAT WERE NOT DETECTED BY SMUT , AND IF SO, CORRECT THEM. IF THE ERROR PERSISTS SEND A FULL (E.G. WITH LISTC .GE. 0, PREFERABLY WITH LISTC .GE. 4) OUTPUT LISTING PLUS PLOT, OR WHATEVER FRAGMENT OF A PLOT WAS MADE, TO DONALD BYRD AT THE ADDRESS ABOVE.

REMARKS ON SPECIFIC MESSAGES...

F01--TO CORRECT THE ERROR, REWRITE THE NOTE (REST) AS SEVERAL TIED NOTES (SUCCESIVE RESTS). (AS FAR AS I KNOW, THIS HAS NEVER OCCURRED.)

F02--THERE ARE 5 POSSIBLE LAYOUT LINE ERRORS, NUMBERED 1-5 AS FOLLOWS.

```
     1  XLN .LT. 12*HS
     2  YDELT .LT. 2*HS
     3  YH+2.*HS .GT. YABMAX, OR
           YMIN .GE. YH            (AT THIS INSTALLATION YABMAX=     )
     4  RESOL .LT. RESMIN          (AT THIS INSTALLATION RESMIN=     )
     5  HS .LT. 15.*RESOL
```

THE NONZERO DIGITS IN THE NUMBER AFTER 'TYPES' INDICATE WHICH ONES ACTUALLY OCCURED.

F07--THE MAXIMUM POSSIBLE NUMBER OF VOICES IN A SCORE IS AN INSTALLATION PARAMETER, IVCMAX. (AT THIS INSTALLATION IVCMAX=   )

F13--IN ORDER TO JUSTIFY THE SCORE, SMUT MAKES A LIST IN EACH BAR (CALLED THE 'SPINE') OF ALL 'TIMES' (E.G., THE BEGINNING OF THE BAR, AN EIGHTH NOTE INTO IT, A TRIPLET QUARTER INTO IT, ETC.) WHEN ANY SYMBOL APPEARS, INCLUDING CLEFS, TIME SIGNATURES, ETC. IF A SCORE IS BEING MADE, THE LIST WILL BE FOR ALL VOICES FOR EACH BAR. IN SOME CASES SEVERAL NONOBVIOUS ENTRIES MAY APPEAR IN THE LIST. NOW, THIS ERROR MESSAGE INDICATES THAT THE MAXIMUM POSSIBLE SIZE OF THE LIST IS BEING EXCEEDED.

F16-- SMUT MUST START A NEW SYSTEM, EITHER BECAUSE OF AN 'S' COMMAND OR BECAUSE THE CURRENT BAR WILL NOT FIT IN THE OLD SYSTEM, BUT IT FINDS THAT THE OLD SYSTEM CONTAINS ZERO BARS. THIS INDICATES (IN THE 'S' COMMAND CASE) TWO 'S' REQUESTS FOR A NEW SYSTEM VERY CLOSE TOGETHER, IN THE OTHER CASE THAT THE CURRENT BAR IS SO LONG THAT IT WILL NOT EVEN FIT IN A WHOLE BY ITSELF. THE LATTER CAN BE CURED BY INCREASING XLN , DECREASING HS (SEE LAYOUT LINE), OR BY USING THE '* HCROWD' COMMAND.

F17--BARLINES MUST COINCIDE IN ALL VOICES. SMUT CHECKS THIS BY COMPARING THE DURATION OF EACH MEASURE IN VOICE 1 TO THE DURATION OF THAT MEASURE IN EACH OTHER VOICE. DURATIONS ARE IN WHOLE NOTES.

N05--EVEN WITH NOSTAFF , SMUT WILL START A NEW LINE WHEN ONE BECOMES FAR TOO CROWDED, I.E. WHEN AT THE END OF A BAR SPACING WOULD HAVE TO BE REDUCED BY A FACTOR OF 1.5 OR MORE TO MAKE THE LINE FIT IN A LENGTH OF XLN MINUS ANY INDENT (SEE THE LAYOUT LINE).

N08--THE KSYM FOLLOWING THE 'TM' MUST BE A LEFT-JUSTIFIED DIGIT, 0-5.

N30--'AREA' HERE MEANS BEAMABLE AREA AS USED IN THE DESCRIPTION OF THE '* BEAM' COMMAND ABOVE, I.E. ONE BEAT UNLESS CHANGED BY '* BEAM'.

W02--SMUT SOMETIMES DOES NOT SPACE NOTES FAR ENOUGH APART TO DRAW NICE LOOKING BEAMS, SLURS, AND GLISSANDI ON OR BETWEEN THEM. BY THE TIME SMUT NOTICES THE PROBLEM AND ISSUES THIS MESSAGE, IT IS TOO LATE TO CORRECT IT.

W04--DECREASING TNOW EVEN SLIGHTLY IS DANGEROUS BECAUSE IT CAN CONFUSE SMUT'S RHYTHM HANDLING AND, FOR EXAMPLE, PRODUCE ERROR S03. IF DECREASING TNOW RESULTS IN A NOTE OR REST HAVING AN ATTACK TIME EARLIER THAN A PREVIOUS NOTE OR REST, ERROR S09 WILL RESULT.

W05--IN TRYING TO DECIDE WHETHER A FRACTIONAL BEAM SHOULD POINT TO THE LEFT OR TO THE RIGHT, SMUT CANNOT FIND A REASONABLE RHYTHMIC STRENGTH FOR THE NOTE WITH THE FRACTIONAL BEAM OR FOR ONE OF THE ADJACENT NOTES.

W06--IN TRYING TO DECIDE WHETHER A FRACTIONAL BEAM SHOULD POINT TO THE LEFT OR TO THE RIGHT, SMUT ATTEMPTS TO GROUP THE FRACTIONAL BEAMED NOTE WITH THE PRECEDING OR FOLLOWING NOTE, BUT FINDS ALL THREE NOTES IN THE SAME GROUP.

EXAMPLES OF ALL ERROR MESSAGES FROM SMUT 2.9

```
NO. ROUTINE MESSAGE
--- ------- ---------------------------------------------------------
F01 DECOMP  COL. 1  RHYTHM TOO COMPLEX FOR DESYNC ROUTINE.
F02 INITSM  LAYOUT LINE ERRORS, TYPES  2040
F03 PASS1   COL. 1  SYNCOPATED ARTIFICIAL GROUP WITH DESYNC IS
            ILLEGAL.
F04 PASS1   COL. 1  ILLEGAL BASIC DURATION FOR ARTIFICIAL GROUP.
            JQP=   0
F05 PASS1   COL. 1  AN ARTIFICIAL GROUP IS ALREADY IN EFFECT.
F06 PASS1   COL. 1  AN S COMMAND WITH KIND=1 MUST PRECEDE ALL
            EXCEPT * COMMANDS
F07 INITVC  TOO MANY VOICES IN SCORE.
F08 WRBUF   PASS1 MEMORY OF  100 COMMANDS PER BAR EXCEEDED.
F09 PASS1   COL. 1  DENOMINATOR OF TIME SIGNATURE IS ILLEGAL.
F10 PASS1   COL. 1  ILLEGAL DURATION.  AFTER TRANSFORMATION NDURB=
            17
F11 PASS1   COL. 1  ILLEGAL NO. OF DOTS.  IDT=  12
F12 PASS3   BAR NO.    17 OVERFLOWS  PASS3  MEMORY OF  2300 WORDS.
            IN VOICE 11
F13 PASS3   BAR NO.    17 EXCEEDS PROGRAM CAPACITY OF  80 ACTION
            TIMES.
F14 INITSM, ZERO BARS OF MUSIC DATA WERE FOUND.
    OVER2
F15 JUSTIF  BEAM TABLE SIZE ( 8) EXCEEDED IN BAR   17 OF VOICE   3
F16 JUSTIF  SYSTEM BEFORE BAR   17 HAS NOTHING IN IT.
F17 JUSTIF  BAR NO.    17 FOR VOICE   3 HAS DURATION    .1250
            (VOICE 1 HAS 1.0000)
F18 PASS1   COL. 1  MULTIBAR REST NOT AT BEGINNING OF A BAR.
            TNOW=   .7500
F19 PASS1   COL. 1  PHYSICAL BARLINE FOLLOWED BY LOGICAL BARLINE
F20 JUSTIF  SHARED STAFF CLEF CONFLICT IN BAR    6 VOICES   2
            AND  3
F21 INITSM  LAYOUT LINE MISSING OR WRONG.


N01 PASS1   COL. 1  ILLEGAL OP CODE 'Z'
N02 PCHCON  COL. 1  PITCH OUT OF BOUNDS.  AFTER TRANSPOSITION
            NOTP=-105
N03 PASS1,  COL. 1  *5 IS AN ILLEGAL CLEF NUMBER
    PROCST
N04 NOTE1   COL. 1  MORE THAN 3 NESTED SLURS REQUESTED.
N05 JUSTIF  SYSTEM BEFORE BAR   28 IS OVERCROWDED BY A FACTOR OF
            1.615
N06 PROCST  COL. 1  ILLEGAL CONTROL VALUE.  NOTP=  8
N07 PROCST  COL. 1  ILLEGAL CONTROL WORD 'SHRAP    '
N08 NOTE4   INVALID TREMOLO SLASH COUNT '4*' IN VOICE   7
N09 SUPER   INVALID NOTE-ASSOCIATED SYMBOL '++' IN VOICE   7
N10 PASS1   COL. 1   -8 IS AN ILLEGAL KEY SIGNATURE.
N11 PASS1   COL. 1  ILLEGAL CHARACTER COUNT OF  33
N12 USERX  UNIMPLEMENTED USER1 CALLED WITH      10      3      4
N13 PROCG   COL. 1  ILLEGAL DURATION FOR GRACE NOTE.  NDURG=    1
```

```
N14 DECODE   COL.  1  ILLEGAL DATA IN NUMERIC FIELD.   IFLAG=-3
             DATA ' 4F'
N15 NOTE1    COL.  1  GLISSANDO STARTS AND ENDS ON SAME PITCH
N16 PASS1    COL.  1  LAST BAR INCOMPLETE.   TNOW= 1.167
N17 PROCT    COL.  1  ILLEGAL OCTAVA PARAMETER  16
N18 PROCT    COL.  1  AN OCTAVA OF  8 IS ALREADY IN EFFECT.
N19 PASS1    COL.  1  INCOMPLETE ARTIFICIAL GROUP AT END-OF-DATA
N20 PASS1    COL.  1   1 UNFINISHED SLURS AT END-OF-DATA
N21 PASS1    COL.  1   28 IS TOO MANY CHARACTERS FOR THIS SIZE STAFF.
N22 NOTE1    NO SLURS ARE IN PROGRESS.
N23 BEAMCK   BEAMING IN BEATS WITH OVER 99 NOTES OR GRACE NOTES IS
             IMPOSSIBLE.
N24 PASS1    CALL TO USER ROUTINE   0 IS ILLEGAL.
N25 INITSM   TOO MANY LINES OF HEADER AND/OR FOOTER
N26 INITSM   CHARACTER SIZE MUST BE A DIGIT.
N27 INITSM   INITIALIZATION OP CODE '7' IS ILLEGAL.
N28 PLTEXT   ILLEGAL JUSTIFICATION TYPE 'D'.  LINE...
             H5 DONALD BYRD'
N29 PLTEXT   NO CLOSING DELIMITER / (OPENING / IN COL.   5)
N30 BEAMCK   MORE THAN  30 BEAMS IN AREA.
N31 PLTEXT   SUBLINE LENGTH OF   8.129 EXCEEDS PAGE WIDTH OF  7.700
             SUBLINE 'SONATA IN D MAJOR, OPUS 10 NO. 3, I, BAR 9-12'
N32 PASS1    UNFINISHED OCTAVA AT END-OF-DATA
N33 INITSM   ILLEGAL CHARACTER 'H' IN BAR OR GROUP LINE
N34 INITSM   GROUPING SPECIFIED FOR OVER 20 VOICES
N35 STRING   STRING BUFFER SIZE EXCEEDED AT CHARACTER 157.  STRING..
             THIS WORK, COMPLETED IN SEPTEMBER 1936, WAS FIRST PERF
N36 INITSM   BARS SPECIFIED FOR OVER 60 SYSTEMS.
N37 PROCST   TNOW OR TLFT CANNOT BE SET TO   8/  0
N38 INITSM   INITIALIZATION OP CODE G IS LEGAL ONLY IN A SCORE.


S01 POSTNR   COL.  1  PROGRAM ERROR***MISSING BARLINE.   DBAR,TNOW,
             DFAC 1.0000 1.2500 1.0000
S02 DECODE   COL.  1  PROGRAM ERROR***BAD FORMAT.   IFLAG=-3
             DATA '(+I3)'
S03 NOTE1,   COL.  1  PROGRAM ERROR***TOO MANY FLAGS. NDURB= 10
    REST1    DLNOTE, DPNOTE       .3333      .0026
S04 JUSTIF   PROGRAM ERROR***    .063 NOT FOUND IN ACTION TIME LIST.
             BAR   17 OF VOICE   3
S05 PASS3    PROGRAM ERROR***OP CODE   6 HAS    5 PARAMETERS
             INSTEAD OF   3
             BAR   17 OF VOICE   3
S06 SLURCK   PROGRAM ERROR***TRIED TO END  2 SLURS BUT ONLY HAD  1
S07 RFILL    PROGRAM ERROR, POINT  12 (Y=5.6480) ABOVE POINT 1
             (Y=5.5712)
S08 RFILL    PROGRAM ERROR,  3 POINTS IS TOO FEW
S09 JUSTIF   PROGRAM ERROR***BAR  27 OF VOICE   4 HAS DECREASING
             TIMES.  250.000 FOLLOWED BY   62.500


W01 RTEST    COL.  1  WARNING, NOTE AND BAR DURATIONS BOTH .GE. 2
W02 PASS4    WARNING.  SLUR LENGTH OF   .075 TOO SHORT IN VOICE  3
W03 THARC    WARNING, IMPOSSIBLE ARC REQUEST.   5.682  8.404  5.441
             8.510   .04
```

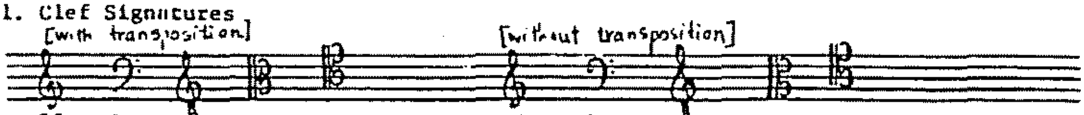
```
W04 PROCST   WARNING, DECREASING TNOW (FROM    0.250 TO    0.063) IS
             DANGEROUS.
W05 LFBEAM   CANT FIND RHYTHMIC STRENGTH FOR FRAC. BEAM DIRECTION,
             0.2794 -99
W06 LFBEAM   ALL THREE NOTES IN GROUP  1 FOR FRAC. BEAM DIRECTION
```

SAMPLE DATA

```
LS   0.005  20            .70
H8 C'SLOTH CANON'
H4 C'FROM THE MUSICAL OFFERING, BWV 1079'
H6 R'J.S.  BACH'
F5 C'SMUT 2.9 SAMPLE PLOT FROM MUSTRAN DATA'
X
* 85HCROWD
*   NOBAR          $*   NODESYNC       =
*   NOCHROM        =
S  1  1 -3  4  4 -2=
P1 4  02 8(ADAGIO)
I   00  0 8INVERSUS
R     2  0         =
R     8  0         =
N 33  0 16  0      =
N 32  0 16  0      =
N 31  0  4  OTI    =
B  1               =
N 31  0  8  0      =
X
* 85HCROWD
*   NOBAR          $*   NODESYNC       =
*   NOCHROM        =
S  1  1 -3  4  4 -2=
P1 4  02 8(ADAGIO)
I   00  011ROYAL THEME
N 29  0  4  OTI    =
N 29  0 16  0      =
N 28 30 16  0      =
N 29  0 16  0      =
N 30  0 16  0      =
N 31  0  4  1      =
N 32  0  8  0      =
B  1               =
N 33  0  4  0      =
X
* 85HCROWD
*   NOBAR          $*   NODESYNC       =
*   NOCHROM        =
S  1  7 -3  4  4 -2=
P1 4  02 8(ADAGIO)
I   00  0 6RECTUS
R     8  0         =
R    16  0         =
N 29  0 32  0      =
N 30  0 32  0      =
N 31  0  8  1      =
N 32  0 16  0      =
N 33  0  4  1      =
R    32  0         =
N 32  0 32  0      =
N 31  0 32  0      =
XX
```

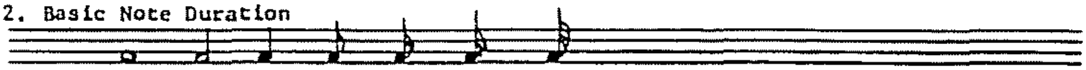
Appendix II

1. Clef Signatures



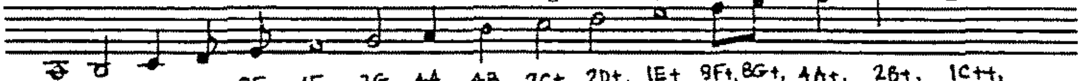
GS, FS. GBS, C3S, C4S, GSZ, FSZ, GBSZ, C3SZ, C4SZ,

2. Basic Note Duration



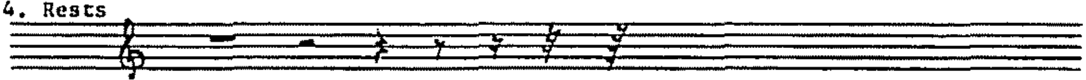
1 2 4 8 16 32 64

3. Note Position (for all clefs not ending in "Z")



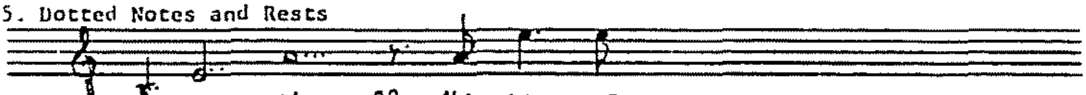
1A-, 2B-, 4C, 8D, 8E, 1F, 2G, 4A, 4B, 2C+, 2D+, 1E+, 8F+, 8G+, 4A+, 2B+, 1C++,

4. Rests



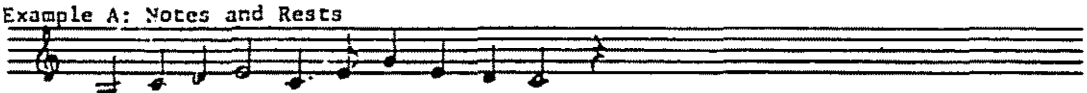
GS, 1R, 2R, 4R, 8R, 16R, 32R, 64R,

5. Dotted Notes and Rests



GBS, 4B-.., 2E.., 1A..., 8R.., 16A, 4E+., 8E+,

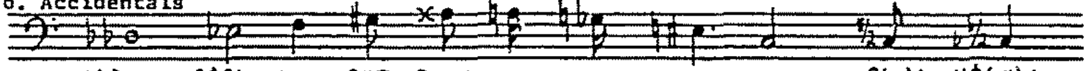
Example A: Notes and Rests



GS, 4G-, 4C, 4D, 2E, 4C., 8E, 4G, 4E, 4D, 2C, 4R,

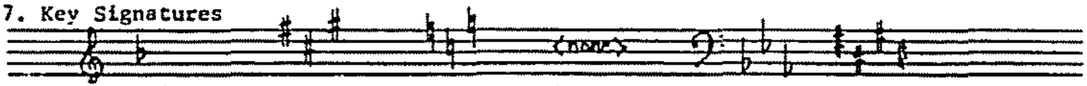
6. Accidentals

6a. Microtones



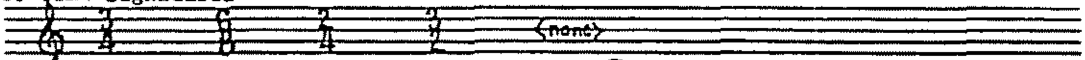
FS, 1$$B, 2$C+, 4D+, 8*E+, 8**F+, 16NF+, 16N$E+, 4N*C+., 2A, 8(+4)A, 4$(-4)A,

7. Key Signatures



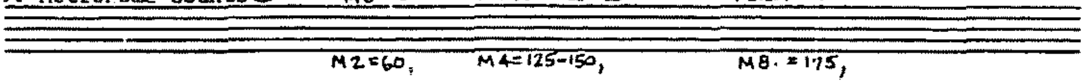
GS, K1$K, K3*K, K3NK, KØK, <zero> FS, K3$K, K4*K,

8. Time Signatures



GBS, 3=4, 6=8, 2=4, 3=2, Ø=4, <zero> ⊛

9. Metrorome Counts ⊛ M𝅗𝅥 = 60 M.M. ♩ = 125-150 M.M. ♪. = 175

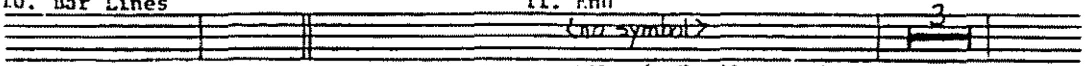


M2=60, M4=125-150, M8.=175,

10. Bar Lines

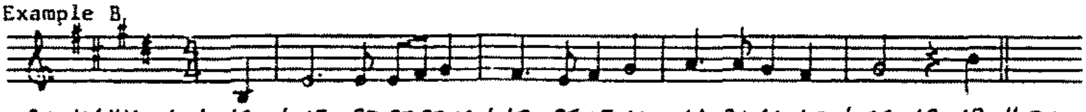
11. End

11a. Long Rests



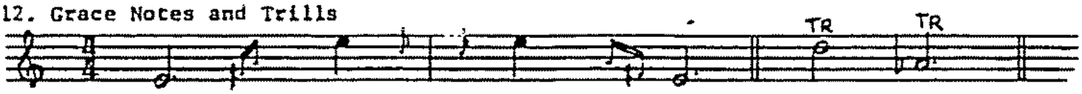
/, //, END (N.B.: NO COMMA) /, LR3, /,

Example B,



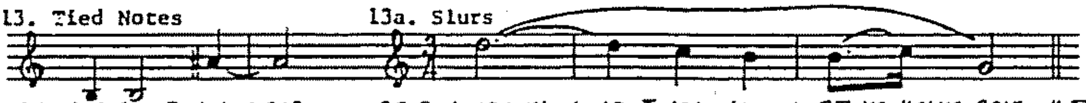
GS, K4*K, 4=4, 4B-, /, 4E., 8E, 8E, 8F, 4G, /, 4F., 8E, 4F, 4G, /, 4A, 8A, 4G, 4F, /, 2G, 4R, 4B, //, END

12. Grace Notes and Trills



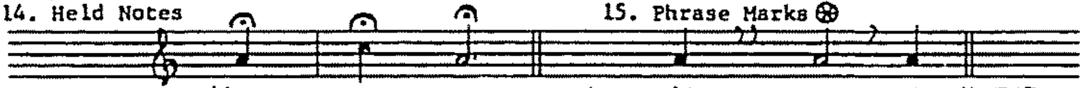
GS, 4=4, 2E, 16P#F, 16P#A, 4E+, 8PC+., /, 16SC+, 4E+, 16SA, 16S#F, 2E., //, 2D+Z1, 2#A.Z1, //, END

13. Tied Notes 13a. Slurs



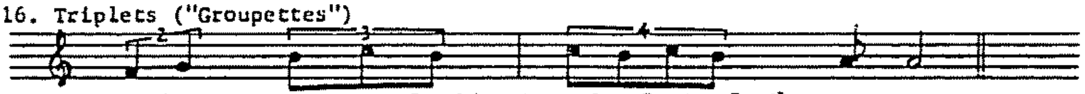
GS, 4B=, 2B-J, 4#A, /, 2AJ, GS, 3=4, 2D+.K1, /, 4D+J, 4C+, 4B, /, 8B.K2, 16C+K2, 2GK1, //, END

14. Held Notes 15. Phrase Marks ⊛



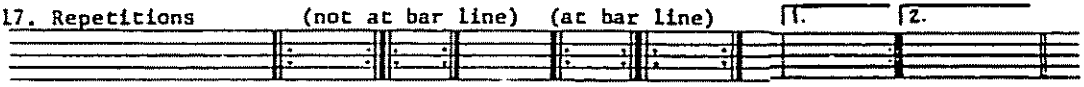
GS, 4AH+H, /, 4C+H+H, 2A.H+H, //, END 4A, ZLL, 2A, ZL, 4A, //, END

16. Triplets ("Groupettes")



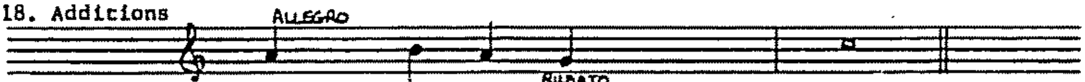
GS, 2(, 4F, 4G, 3), 3(, 8B, 8C+, 8B, 3), /, 4C, 8C+, 8B, 8C+, 8B, 3), 8A, 2A, //, END

17. Repetitions (not at bar line) (at bar line)



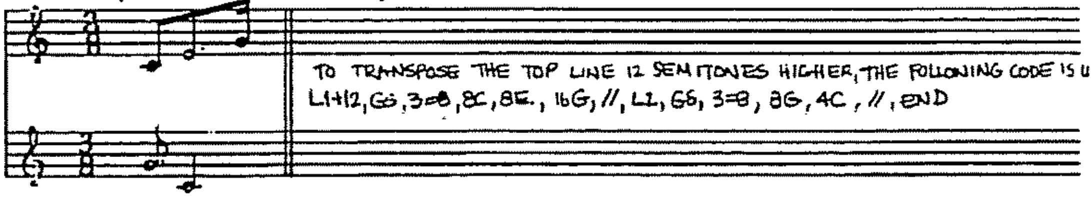
T(, T)/(, T), T/(, T)/(, T)/, /, T1), T), T2), //, END

18. Additions



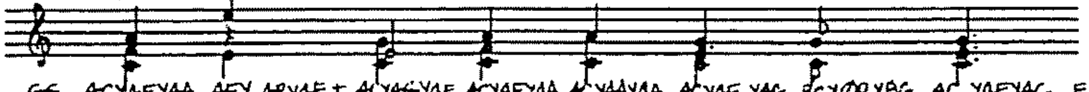
GS, 'ALLEGRO', 4A, 4B, 4A, 'RUBATO'==B-, 4G, /, 1C+, //, END

19. Multiple Staffs and Transposition



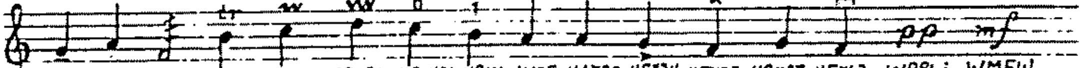
TO TRANSPOSE THE TOP LINE 12 SEMITONES HIGHER, THE FOLLOWING CODE IS USED:
L1+12, GS, 3=8, 8C, 8E., 16G, //, L1, GS, 3=8, 8G, 4C, //, END

20. Multiple Voices on a Single Staff ⊛



GS, 4CY4FY4A, 4EY ARY4E+, 4CY4GY4E, 4CY4FY4A, 4CY4AY4A, 4CY4E.Y4G, 8CYØRY8G, 4C.Y4EY4G., END

21. Performance Indications and Dynamics



GS, 4GV1, 4AV3, 2FV7, 4BZ1, 4C+Z3, 4D+Z4, 4C+X31, 4BX1, 4AX5, 4AX20, 4GX24, 4FX25, 4GX27, 4FX62. WPPW, WMFW,

Index to Definitions

N.B. All of the following are *music* terms, unless indicated otherwise. The list is not complete: many more terms are defined in Chapter 2.

Credits

Grateful acknowledgement is made to the following publishers for permission to quote excerpts from the following material.

Bartók: *String Quartet No. 4*: copyright © 1929 by Universal Edition; renewed 1956. Copyright and renewal assigned to Boosey and Hawkes, Inc. Reproduced by permission.

Berg: *Violin Concerto*: copyright © 1936 by Universal Edition; copyright renewed 1964. All rights reserved. Used by permission of European-American Music Distributing Corp., sole U.S. agent for Universal Edition.

Debussy: *Preludes for Piano, Book I*: copyright © 1910 Durand S.A. Used by permission of the publisher, Theodore Presser Co.

Debussy: *Suite "Pour le Piano"*: copyright © 1901 Durand S.A. Used by permission of the publisher, Theodore Presser Co.

Ives: *Three Places in New England*: copyright © 1935 and 1976, Mercury Music Corp. Used by permission.

Ravel: *Le Tombeau de Couperin*: copyright © 1918 Durand S.A. Editions Musicales and Editions Arima joint publication. Used by permission of the publisher, Theodore Presser Co., sole representative, U.S. and Canada.

Schoenberg: *Pierrot Lunaire*: copyright © 1914 by Universal Edition; renewed 1941 by Arnold Schoenberg. Used by permission of Belmont Music Publishers.

Scriabin: *Sonata No. 1* and *Sonata No. 7*: copyright © 1976 by International Music Co. Used by permission of International Music Co.

Vita

Donald Alvin Byrd was born in Chicago on September 4, 1946. Starting in 1964, he attended Indiana University in Bloomington, where he studied music composition with Bernhard Heiden and Roque Cordero, receiving the degree of Bachelor of Music (in composition) in 1968. He received the degree of Master of Science in Computer Science from Indiana University in 1975. From 1969 to 1976 he was an applications systems programmer at the Wrubel Computing Center of Indiana University. During this period he was also a programmer and laser artist for the Soleil Laser Music Spectacle. In 1977 he was briefly employed by Norlin Music Corporation of Chicago as a software engineer, after which he returned to the Wrubel Computing Center. Since April 1983 he has been a software engineer with Kurzweil Music Systems, Boston.

Byrd is a member of the Boards of Directors of the Computer Music Association and of the New England Computer Music Association.